_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Evaluación subjetiva de un sistema de control de sonido multiusuario

Sound Challenge

Autora: Lidia Martínez Salvador

Tutora: María de Diego Antón

Cotutores: Francisco J. Martínez Zaldívar y Laura Fuster Criado

Trabajo Fin de Máster presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Máster en Ingeniería de Telecomunicación

Curso 2016-17

Valencia, 3 de julio de 2017

Resumen

En esta memoria se describe el diseño e implementación de una herramienta web que pretende realizar una evaluación subjetiva de los sonidos. En la primera parte se describe la explicación de los métodos utilizados para su diseño, continuando con la descripción de la implementación. La aplicación de sonido que se tiene como referencia, persigue el confort acústico de usuarios colocados en diferentes posiciones de una sala. Para conseguir la eficiencia de la aplicación se requiere de una evaluación subjetiva para validar el funcionamiento.

Se realizan pruebas de escucha basadas en comparación de pares teniendo como finalidad evaluar los sonidos presentándolos en pareja. El sujeto debe indicar su preferencia hacia uno de los dos sonidos teniendo en cuenta la característica subjetiva que se indique. El objetivo es conseguir la mayor cantidad de datos en el menor tiempo posible, por ello se ha implementado una aplicación web con el fin de obtenerlos de cualquier parte del mundo y de forma inmediata. Dicha aplicación web se ha desarrollado mediante el *framework* Django, donde se ha realizado un trabajo *full-stack* programando tanto la parte de usuario con HTML5, CSS3 y JavaScript, como la parte del servidor gracias a Django mediante Python. El motor de base de datos utilizado es un módulo implementado de Python llamado *SQLite3*.

La aplicación web está orientada a usuarios no expertos en audio, por ello las pruebas tienen una apariencia austera y sencilla, además de ser fácil de usar. Posee un sistema de registro mediante el cual los usuarios pueden acceder. Aparecerán diferentes opciones dependiendo si se es administrador o no, estando preparada en modo administrador para quien vaya a diseñar, modificar y analizar las pruebas. Todo ello teniendo posibilidad de ejecutarla en cualquier plataforma conectada a internet gracias a su diseño adaptativo. La aplicación se denomina *Sound Challenge*.

Resum

En aquesta memòria es descriu el disseny e implementació de una ferramenta web que pretén realitzar una avaluació subjectiva del sons. En la primera part es descriu la explicació dels mètodes utilitzats per al seu disseny, continuant amb la descripció de la seua. L'aplicació de so la qual es té con referencia, persegueix el confort acústic d'usuaris col·locats en diferents posicions d'una sala. Per a aconseguir l'eficiència de la aplicació es requereix d'una avaluació subjectiva per a validar el funcionament.

Es realitzen probes d'escolta basades en comparació de parelles amb la finalitat d'avaluar cada parella. El subjecte té que indicar la seua preferència cap a un dels dos sons tenint en compte la característica subjectiva que s'indiqui. L'objectiu es aconseguir la major quantitat de dades en el menor temps possible, per això s'ha implementat una aplicació web amb la finalitat d'obtenir dades de qualsevol part del mon i de forma immediata. Aquesta aplicació web s'ha desenvolupat mitjançant el *framework* Django, on s'ha realitzat un treball *full-stack* programant tant la part d'usuari amb HTML5, CSS3 i JavaScript, com la part del servidor gràcies a Django mitjançant Python. El motor de base de dades utilitzat es un mòdul implementat de Python cridat SQLite3.

L'aplicació web està orientada a usuaris no experts en àudio, per això les probes tenen un semblant auster i senzill, a més es fàcil d'usar. Posseeix un sistema de registre mitjançant el qual els usuaris poden accedir. Hi han diferents opcions depenent si se es administrador o no, estant preparada en mode administrador per qui vaja a dissenyar, modificar i analitzar les probes. Tot això tenint la possibilitat de executar-ho en qualsevol plataforma connectada a la xarxa gràcies a el seu disseny adaptatiu. L'aplicació es denomina *Sound Challenge*.

Abstract

In this paper is described the design and implementation of a web tool which its objective is performed subjective evaluation of sounds. On the first part it is described the explanation of the used methods for its design, continuing with its implementation. The sound application, which is had as reference, pursues the user acoustic comfort placed in different positions in a room. To achieve the application efficiency it is necessary make a subjective evaluation to validate its quality.

Listening tests are preformed based on paired comparisons with the aim of evaluating each pair. The user must set his preference to one of the two sounds regarding a subjective characteristic. The objective is to achieve the greatest amount of data and for that reason a web application has been implemented to get data from any part of the world and in an immediate way. This web application is developed through Django framework, where a full-stack work has been made by programming both user part with HTML5, CSS3 and JavaScript, and server part thanks to Django with Python. The engine data base used is a python module named SQLite3.

The web application is thought for non-experts users in audio, for that purpose the tests have austere and simple appearance, and also it has a friendly usage. The app has a sign up system where the users can sign in on it. Different options appear depending on if the users has administrator permission or not, where the administrator mode is ready for the user who is able to design, modify and analyse the tests. Moreover the app is available in any electronic device with internet connectivity because it has a responsive web design. The application is named *Sound Challenge*.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1. Introducción

Este capítulo presenta una breve introducción del trabajo realizado resaltando los elementos más relevantes para la posterior explicación.

Por otra parte, se destaca la principal motivación razonando por tanto, la decisión de realizar este trabajo.

Capítulo 1. Introducción

Este Trabajo Fin de Máster (TFM) ha sido desarrollado con el fin de avanzar y estar más cerca de la comprensión de uno de los temas más complicados que se producen en la ciencia, la opinión subjetiva del ser humano. En concreto dicho trabajo es parte de un proyecto de investigación con título *D-NOISE: Distributed Network of Active Noise Equalizers for Multi-user Sound control,* el cual pretende realizar un sistema que permita obtener el campo acústico deseado en torno a cada usuario, siendo el objetivo del presente TFM evaluar la aceptación de las personas frente a la aplicación del mismo. El método que se ha llevado para realizar los diferentes test subjetivos de audio ha sido la comparación de pares de sonidos resultantes del sistema de ecualización, donde se compara dos audios que a priori la persona que va a evaluarlos, no sabe cuáles son.

Como se ve a lo largo del trabajo, una de las partes claves del mismo fue el medio en el que se quería implementar dichos tests, ya que se deseaba que fuera accesible al mayor número de personas y a la vez que fuera fiable y seguro con respecto a los datos y los usuarios. En concreto, se propuso realizar la herramienta en un entorno web y por ello, como se comprueba más adelante, la mayor parte del trabajo trata de la propuesta y resolución de las diferentes partes que tiene la aplicación web y de las herramientas con las que ha sido desarrollada.

Antes de comenzar a explicar cualquier concepto con respecto a la evaluación subjetiva o al desarrollo de la aplicación, es conveniente explicar de forma adecuada el porqué de la creación de este trabajo.

1.1. *Motivación*

La principal razón del presente documento es la actual necesidad de conocer la percepción acústica subjetiva del usuario de un sistema de ecualización de ruido activo que está actualmente en desarrollo. Este sistema está pensado para mejorar el bienestar y confort acústico de las personas en un entorno ruidoso. Por tanto, como actualmente está en proceso de desarrollo, es oportuno a la par que necesario realizar un estudio capaz de evaluar las preferencias y opiniones de las personas sobre el campo acústico generado con el sistema implementado. Dicho estudio, se define como evaluación subjetiva, ya que pretende conseguir la opinión de las personas la cual es dispar de una a otra. Para ello, se implementan las denominadas pruebas de escucha.

Normalmente cuando se desea realizar los test de escucha, se realizan de forma local y con medios fijos para asegurar de alguna manera los resultados y tenerlo controlado, pero este tipo de práctica suele ser costosa, lenta y además no se suelen obtener gran cantidad de datos, ya que hay que reclutar gente de forma física para que realicen las pruebas. Por ello, otro de los motivos por los que surgió este trabajo, es la implementación de los test en una plataforma capaz de solventar esos problemas, un lugar donde fuera posible la realización de pruebas de escucha fácilmente accesible para cualquiera: una aplicación web.

Este formato tiene ventajas pero también existe algún inconveniente: no es posible un control total de todas las variables, y esto provoca que los resultados no sean a priori totalmente verídicos, aunque se recolecten muchos datos del usuario y sus medios, siempre existirá error. Pero al ser una herramienta que se accede por internet, la idea es que accedan de todas las partes del mundo y así conseguir gran cantidad de datos. Por tanto si se tienen más resultados y se piden la mayor cantidad de datos al usuario sin saturarlo, se puede solventar el problema del error en el análisis. Por otra parte, en la realización de las pruebas no sería adecuado recolectar todos los resultados sin tener ningún tipo de filtro, como se verá más adelante, se utilizan diferentes

parámetros estadísticos para poder excluir o no a las personas que tengan resultados coherentes con la prueba. En concreto, las pruebas de escucha están basadas en las comparaciones de pares las cuales se explicarán con mayor detalle más adelante.

Con estas dos ideas, podemos afirmar que finalmente podremos tener unos resultados bastante aceptables además que realmente será accesible a todo el mundo ya que el idioma utilizado ha sido el inglés en su totalidad. Por otro lado, la herramienta web ha sido desarrollada con el fin de ampliarse en un futuro teniendo como base dicho objetivo de exclusión y tratamiento de datos.

Por tanto uno de los principales desafíos al realizar la deseada aplicación web fue el crearla desde cero, ya que al principio no se tenían los conocimientos adecuados para crear un sistema modelo-vista-controlador (Capítulo 3). Para realizar más asequible ese escalón, se optó por utilizar el *framework* Django. Permite de manera relativamente rápida y sencilla la creación de un sistema web completo, haciéndose cargo el mismo *framework* de los temas de seguridad. Más adelante se detallan las diferentes partes más importantes de Django que se utilizan para crear la aplicación web de evaluación subjetiva bautizada con el nombre de *Sound Challenge*.

Para finalizar la explicación de la existencia de *Sound Challenge*, hay que destacar de que aparte de toda la teoría sobre los test subjetivos, los cálculos estadísticos detrás de ello y los desafíos de programación que hay que hacer frente, uno de los puntos más importantes para que un proyecto como éste salga con éxito, es que sea vistoso, que a la gente le llame la atención hacerlo ya que el objetivo es que lo haga todo el mundo, de cualquier parte del mundo y de cualquier condición. Por ello, un punto bastante importante fue el diseño, comodidad y sencillez de la aplicación, como se diría en terminología anglosajona, debía tener un *friendly use*. Por ello otra de las dificultades añadidas fue realizar un diseño adecuado además de que pudiera utilizarse en cualquier dispositivo que se conectara a internet.

Capítulo 2. Objetivos y estructuración

Con el presente apartado se pretenden exponer los objetivos planteados en el trabajo. De esta forma, se comprenderá el porqué de cada fase abordada.

Además en este mismo capítulo se muestran de manera estructurada cada paso que se ha ido tomando en la evolución del proyecto, mostrando su organización y estructuración en el tiempo mediante un diagrama de Gantt.

Capítulo 2. Objetivos y estructuración

2.1. *Objetivos*

La finalidad de este trabajo fue diseñar y desarrollar una aplicación web capaz de evaluar la sensación subjetiva de diferentes sonidos generados por el sistema de control de campo sonoro. Si bien esta herramienta puede emplearse para evaluar cualquier tipo de sonidos, no sólo los que han motivado el trabajo. Como punto de partida, la aplicación web se basaría en un software de test de comparación de parejas ya realizado en Matlab por componentes del grupo de investigación GTAC (Grupo de Tratamiento de Audio y Comunicaciones) perteneciente al iTEAM (Instituto de Telecomunicaciones y Aplicaciones Multimedia) de la UPV (Universitat Politècnica de València). Para conseguir realizar la aplicación web, se plantearon los siguientes objetivos:

Con respecto a la parte previa y teórica:

- Comprensión de las partes básicas del sistema de ecualización de control de ruido activo.
- Estudio del software ya existente sobre la evaluación subjetiva basada en comparaciones de parejas realizado en Matlab.
- Mejora del software de Matlab creado para adaptarlo a las características de las señales acústicas que se iban a evaluar.
- Estudio de las diferentes partes y características que conlleva realizar un estudio subjetivo del sonido, haciendo real hincapié a los métodos de análisis utilizados, además de otros para la futura mejora de la obtención de datos estadísticos.
- Selección y preparación de la naturaleza de los sonidos a comparar para conocer qué tipo de características tienen y así adaptar la aplicación web.

Con respecto al diseño previo y el estudio de las herramientas de programación de la aplicación web:

- Estudiar cómo funciona el *framework* Django y así comprender desde el principio qué partes tiene haciendo hincapié en comprender cómo se administra la base de datos la cual se importa mediante un módulo de *Python* denominado *SQLite3*.
- Investigar a lo largo del desarrollo los distintos lenguajes empleados (*Python, HTML, JavaScript, jQuery* y *CSS3*) para realizar una programación adecuada desde principio a fin.
- Para realizar un diseño adecuado y llamativo, investigar la opción de utilizar la librería denominada *Polymer*.
- Realizar la aplicación web adaptable a cualquier dispositivo de hoy en día (smartphone, tablet, PC). Por ello, investigar el uso del *framework* Bootstrap e implementarlo en la programación.
- Investigar y realizar un diseño previo adecuado para la realización de las pruebas de escucha.

Con respecto al desarrollo e implementación de la aplicación web:

- Creación de un sistema de registro cumpliendo las normas de protección de datos adecuadas.
- Sistema central de la aplicación donde los usuarios registrados puedan acceder a las diferentes pruebas de la manera más sencilla posible.
- En el test, adquirir más datos del usuario como qué tipo de auriculares usará y el tipo de tarjeta de sonido que posee. Esto es para adquirir más datos sin abrumar al usuario.
- A continuación un sistema de entrenamiento para que el usuario conozca las mecánicas del test. Al finalizar el test, una pequeña realimentación de los resultados obtenidos para motivar al usuario.
- Implementar un registro de tipo administrador para acceder a la creación y análisis de las pruebas realizadas. Por tanto, realizar de forma íntegra la creación, modificación y resolución de las pruebas.
- Administración correcta de la base de datos teniendo en cuenta todos los detalles de seguridad.

2.2. *Estructuración*

2.2.1. Estructura de la memoria

Para guiar al lector a través del presente trabajo, a continuación se describe la estructura del documento de forma breve:

Capítulo 1 - Introducción

Se introduce el proyecto destacando los elementos más relevantes del mismo. Además se razona de forma concisa por qué se decidió y necesitó realizar el estudio subjetivo del audio.

Capítulo 2 – Objetivos y estructuración

Es el capítulo en el que nos encontramos. Con él, se pretende enmarcar los objetivos del TFM además de mostrar de forma clara la estructura y organización que se ha llevado para conseguir realizarlo.

Capítulo 3 – Marco teórico

Se explican todos los conceptos sobre la evaluación subjetiva relevante para el proyecto. Abarcando desde una explicación breve del sistema de procesado a evaluar, hasta las características de la evaluación subjetiva con los datos aportados del análisis final.

Capítulo 4 – Herramientas de desarrollo

Se explica de forma más concisa posible, las herramientas utilizadas y sus características, con la finalidad de que el lector comprenda por qué fases ha pasado el proyecto y qué se ha tenido que utilizar.

Capítulo 5 – Desarrollo de la aplicación web

Este sería el núcleo del proyecto. Se describe de forma más esquematizada posible las diferentes partes de la aplicación web enfocándolo a una estructura Modelo-Vista-Controlador (MVC). Se presenta un esquema de bloques general y a medida que nos adentramos en el capítulo se va seccionando para explicar cada sub-bloque. Terminando con la explicación de los resultados obtenidos por la aplicación mediante un ejemplo.

Capítulo 6 – Conclusiones y líneas futuras

Finalmente se medita sobre los resultados del proyecto y si se han conseguido los objetivos planteados, además de proponer diferentes líneas futuras para la mejora y mantenimiento de la aplicación web.

2.2.2. Distribución de tareas

Es importante resaltar qué fases ha requerido el desarrollo del proyecto para entender el proceso llevado a cabo y las dificultades encontradas. Además la siguiente descripción de las tareas permite la correcta interpretación del diagrama temporal que aparece en el siguiente punto. Las tareas se dividen en las siguientes fases:

I. Fase de trabajo previo:

- Analizar el software ya creado de evaluación subjetiva de Matlab.
- Mejorar dicho software y así comprender su funcionamiento y qué requisitos deberá tener la aplicación web.
- Determinar qué características se desea que tenga la aplicación web para que cumpla su fin con las preferencias del personal de GTAC. Así pues, realizar un estudio sobre diferentes ejemplos de pruebas de escucha.
- Realizar el estudio y la práctica necesaria para poder dominar de manera básica las funcionalidades de Django y así comprender su estructura.

II. Fase de búsqueda bibliográfica:

- Investigación de los diferentes estudios realizados en torno al desarrollo del sistema de ecualización de ruido activo.
- Realizar un estudio sobre la creación de pruebas subjetivas de forma concreta con respecto a las pruebas de escucha y la comparación de parejas.
- Estudio y búsqueda de buenas prácticas sobre programación de páginas web para cada funcionalidad de la misma, siendo así estudiado todos los lenguajes de programación implicados (*HTML, Python, JavaScript, CSS3, biblioteca jQuery*).
- Estudio de las funcionalidades de Django y sus diferentes librerías. Conocer la administración correcta de todos los controles necesarios para realizar la base de datos, su modificación y acceso, tanto directamente desde Django como desde la página web con *JavaScript.* Estudio de la utilización de *cookies* correcta.
- Estudio de tanto la librería denominada *Polymer* como el *framework Bootstrap* para conseguir realizar una programación adecuada.

III. Fase de análisis y verificación de requisitos:

- Partiendo de los requisitos necesarios del trabajo previo proporcionados por el personal investigador del GTAC, realizar un primer esbozo y estructura de la aplicación web, incluyendo todas las partes que debe tener.
- A lo largo del desarrollo, ir consultando si las características y formato realizado es el adecuado, con el motivo de modificar o añadir cualquier característica extra que se haya pensado.

IV. Fase de desarrollo:

- Creación de la pantalla principal para poder registrarse o si ya se está registrado, iniciar sesión en la aplicación.
- Creación del formulario básico indicando toda la información necesaria para el usuario (añadido el modelo/tabla del formulario).
- Creación de un primer diseño de la página principal de la aplicación web con las diferentes pestañas como opciones.
- Creación del sistema de diseño de *tests* (añadido el modelo/tabla de las señales de audio, el modelo/tabla de los diseños y el modelo/tabla de los usuarios que han realizado el test).
- Creación de la vista de adquisición de datos del usuario al realizar una prueba. Y creación de la vista de entrenamiento del usuario frente a las pruebas de escucha.
- Creación de una vista de *feedback* al terminar cada prueba.
- Creación de un sistema de modificación de pruebas sin testear.
- Creación del sistema de análisis de forma gráfica importando toda la programación de análisis estadístico en Matlab a lenguaje *JavaScript*.

V. Fase de corrección y evaluación:

- Cambiar formato entero de la aplicación para que fuera adaptativo, es decir, emplear el *framework* Bootstrap.
- Cambiar y mejorar detalles de seguridad.
- Evaluar las respuestas a nivel de usuario y los resultados de las pruebas.
- Analizar necesidades futuras para el mantenimiento y mejora de la aplicación.

VI. Fase de redacción:

- Participación en el artículo "*TESTSOUND: A Matlab App for the Psychoacoustic Evaluation of Sounds by means of Paired Comparisons*" con autores G. Piñero, L. Fuster, M. Ferrer, L. Martínez, M. de Diego, A. González., siendo aceptado en el número 9 de la revista *Waves* [1] que se publicará en noviembre del 2017.
- Redacción de la presente memoria.

2.2.3. Diagrama de Gantt

A continuación se detalla el diagrama de Gantt que muestra de forma temporal en qué momento de este año 2017 se han desarrollado las diferentes tareas anteriormente descritas:

	FEBRERO				MARZO				ABRIL				MAYO				JUNIO			
Semana	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Fase I. Trabajo previo		**X**	**X**	**X**	**X**															
Analizar el software de Matlab		X	X																	
Mejorar el software de Matlab			X	X																
Concretar características				X	X															
Estudio y práctica de Django				X	X															
Fase II. Búsqueda bibliográfica		**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
Fase III. Requisitos		**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**		
Realizar casos de uso					X	X	X	X	X											
Verificación y modificación de elementos					X	X	X	X	X	X	X	X	X	X	X	X	X	X		
Fase IV. Desarrollo					**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**					
Pantalla principal					X	X	X	X	X	X	X	X	X	X	X					
Formulario						X	X	X	X	X	X	X	X	X	X					
Primer diseño del *main*							X	X	X	X	X	X	X	X	X					
Diseño y creación de pruebas de escucha							X	X	X	X	X	X	X	X	X					
Configuración de test de usuario								X	X	X	X	X	X	X	X					
Ejecución de la prueba de escucha								X	X	X	X	X	X	X	X					
Vista de *feedback*										X	X	X	X	X	X					
Modificación de pruebas sin testear												X	X	X	X					
Sistema de análisis														X	X					
Fase V. Corrección y evaluación																**X**	**X**	**X**	**X**	
Cambio del formato entero con *Bootstrap*																X	X	X		
Mejora de detalles de seguridad																		X	X	
Evaluación a nivel de usuario y resultados																	X	X		
Análisis de necesidades futuras																		X		
Fase VI. Redacción de la memoria															**X**				**X**	**X**

Tabla 1: Diagrama temporal del TFM

Capítulo 3. Marco teórico

En este capítulo se explica los fundamentos teóricos aplicados para la realización y evaluación de los test de pares, centrándose en los conceptos más relevantes utilizados en la aplicación de *Sound Challenge*. Además se introduce ligeramente la funcionalidad del sistema de ecualización.

Capítulo 3. Marco teórico

3.1. *Introducción al sistema de ecualización*

El presente trabajo, como se ha comentado en la introducción, tiene como cometido el evaluar de forma subjetiva los audios resultantes de un sistema multiusuario de ecualización de ruido. Este sistema tiene como objetivo crear un campo acústico deseado conservando parte del ruido residual con un perfil determinado. Previamente al sistema de ecualización cabe definir un sistema de cancelación, el cual está diseñado para cancelar en ciertos lugares del espacio un ruido que a priori puede resultar molesto, utilizando para ello, ondas sonoras de misma amplitud pero con diferente fase. En concreto, para el proyecto en el que se engloba esta aplicación, han sido generadas utilizando un algoritmo adaptativo basado en el filtrado-X LMS [2]. La diferencia de un sistema de cancelación [3] respecto a un sistema de ecualización es que este último pretende actuar sobre las distintas frecuencias del ruido para amplificarlas o atenuarlas conformando así una respuesta o espectro frecuencial final específico de cada usuario atendiendo a sus necesidades o preferencias. Por tanto, un sistema de ecualización se puede ver como un sistema general en el que el sistema de cancelación corresponde a la atenuación total de todas las frecuencias. Toda la información relevante con respecto al diseño de los algoritmos empleados se puede encontrar en [2].

En concreto y como primera aplicación, se quiere realizar la ecualización de ruidos periódicos porque son inherentes a los ruidos generados por el motor de un vehículo. El escenario dónde actúa es un entorno cerrado en el que varios usuarios situados en posiciones diferentes de la sala escogen el perfil espectral deseado, conformando una respuesta frecuencial única a cada usuario que permite que se escuchen sonidos diferentes en distintas zonas del espacio, no solo su cancelación o silencio [4]. A pesar de que hoy en día está en proceso de desarrollo, existe un prototipo del futuro sistema que se desea implementar en el interior de los vehículos (figura 1) y con el cual se grabarán los audios resultantes que se usarán en la evaluación subjetiva final.

Fig. 1: Demostrador del sistema realizado en el GTAC (iTEAM, en la UPV)

En la figura 1 se puede observar que el sistema que compone cada asiento del vehículo está compuesto por un par de micrófonos a la altura de los oídos y un par de altavoces direccionados a estos. Éste sería un prototipo inicial, donde en un futuro se podrían integrar todos los componentes en el mismo vehículo. Las zonas de silencio o ecualización implementadas se encuentran dónde están los micrófonos, es decir, en la zona donde se puede controlar el sonido con los micrófonos de error.

Audios resultantes para su posterior evaluación, es importante haberlos grabado de acuerdo a una referencia válida capaz de simular lo que percibiría una persona. Para estos casos se utiliza un maniquí de *Head acoustics*^{TM} el cual representa un sistema binaural teniendo una respuesta estándar de la escucha humana (es decir, incluyendo el efecto de la cabeza y orejas en la grabación del sonido). Esta misma herramienta se utilizó para la evaluación de un sistema de cancelación activo de ruido distinto al que nos enfrentamos ahora [3].

Una vez se consiguen los audios del sonido procesado con diferentes perfiles de ecualización, el siguiente paso es averiguar la percepción de los futuros usuarios para conocer de primera mano qué nivel de confort se percibe y si realmente la ecualización es óptima. Para ello hay que aplicar un método de evaluación subjetiva que se verá a continuación, pero antes de nada para elegir y diseñar un método correcto, hay que tener claro si los sujetos que van a realizar las pruebas subjetivas son expertos o no en la misma. Para este trabajo se ha elegido un método el cual está pensado para que lo realicen personas no expertas, es decir, que lo pueda realizar cualquier persona.

La gran diferencia en que esté destinado a un grupo experto o no, es que para un grupo no experto es importante mantener la atención del mismo y evitar que la prueba se convierta en tediosa o aburrida, ya que en principio no tiene intención de realizarla, sino que se le habría reclutado. Por ello los elementos con los que deberán interactuar deben ser lo más simples y concisos posible.

El tipo de evaluación subjetiva está basado en pruebas de escucha, consiguiendo con ellas que el jurado pueda dar su opinión con respecto a la percepción de calidad que tiene de los audios, entendiendo como calidad el nivel de confort que produce el sistema al ecualizar los sonidos ruidosos, es decir, que 'suene bien' para el jurado.

3.2. *Método de la comparación de parejas*

Existen diferentes métodos para evaluar de forma subjetiva. Dependiendo del tipo de prueba que se vaya a realizar y qué tipo de audio se evaluará, se utiliza uno u otro. Por ejemplo para evaluar la calidad de los diferentes audio-codecs se utiliza el método MUSHRA (*MUltiple Stimuli with Hidden Reference and Anchor*) definido en la recomendación ITU-R BS.1534-3. En concreto, este trabajo no se centra en evaluar alguna característica de calidad del sonido de forma técnica, sino evaluar parámetros subjetivos. Existen ciertos estudios como en [5] donde evalúan pruebas de escucha con el método MUSHRA, pero otros estudios donde no se es posible cumplir con las características del método, por ejemplo que los sujetos no sean expertos, se optan por otros métodos como el de comparación de parejas.

Con dicho método se consigue que el jurado no tenga que conocer en concreto el tipo de audio el cual va a evaluar, ya que solamente tendrá que elegir entre un audio u otro. En cada paso del test se muestra una pareja de audios donde el juez deberá elegir uno de ellos dependiendo de un parámetro subjetivo en concreto (preferencia, molestia…), estando el juez en desconocimiento sobre qué audio es cual ya que están identificados como 'Audio A' y 'Audio B'.

Este tipo de estudios normalmente se aplican a la industria automovilística donde es interesante conocer las sensaciones que produce el sonido que emite un automóvil para el usuario final [6]. En algunos, se utiliza en vez de la elección entre un sonido u otro, se aplica una escala del 1 al 10, pero para los sujetos no expertos este tipo de escalado puede ser confuso y por tanto no se obtendrían datos fiables. Además, si resulta que se desea evaluar una gran cantidad de audios en una misma prueba y/o varios parámetros, el método de la comparación de parejas resulta más liviano y menos tedioso para el jurado, aparte de que el análisis estadístico posterior resulta ser más sólido dando unos valores representativos de la realidad.

No sólo se ha utilizado para la industria automovilística, este método se puede utilizar para cualquier situación donde se desee evaluar características subjetivas sin tener a disposición jurado experto. Por ejemplo en [7] se estudió las preferencias de niños entre 3 y 10 años frente a los sonidos que emitían los juguetes adaptados a su edad. Con el método de las parejas se consiguió averiguar que existía distinción entre las preferencias de niños y niñas. La aplicación *Sound Challenge* desarrollada en este trabajo, está basada en el método que siguieron los investigadores del mismo documento [7], ya que además para probar todas las funcionalidades y dejar preparada la aplicación para las pruebas subjetivas, se han utilizado los mismos audios empleados en dicho estudio. Ya que a día de hoy, no se tienen los resultados de las pruebas de escucha con respecto al sistema multiusuario de ecualización de sonido, que es una propuesta de futuro.

Antes de explicar qué algoritmos y procedimientos se utilizan para hacer el análisis de las pruebas, destacar que la mayoría de las pruebas subjetivas comentadas se tuvieron que realizar en un laboratorio, controlando eso sí, todos los elementos que pertenecen a la prueba y seleccionando premeditadamente los sujetos que la realizaron. Este tipo de procedimiento puede asegurar la fiabilidad de los resultados pero suelen ser lentas y tediosas para los investigadores, además de estar limitadas por no poder reclutar a muchos sujetos. Por ello uno de los retos comentados al principio del documento, fue de trasladar las pruebas subjetivas de comparación de parejas a una plataforma web ya que permite una adquisición de datos mucho más rápida y fácil de analizar, además de que se tiene un alcance mayor de jurados, pudiendo llegar a ser global. Este tipo de implementación no es nueva, otros estudios de audio se han realizado mediante plataforma web resultando exitosos [5]. Tiene ciertas desventajas, como la probabilidad de error que esta genera por el no poder controlar los elementos de la prueba de escucha. En concreto los elementos a emplear son un dispositivo que reproduzca el audio (tarjeta de sonido), unos auriculares y las características de la persona. Todo ello se tiene en cuenta en *Sound Challenge*, como se ve más adelante, los usuarios introducen la configuración del *hardware* del test además de sus datos de edad y género como requerido, teniendo así el control más apropiado para ser una aplicación web. Con ello y con un proceso de selección definiendo un parámetro para excluir jurados no válidos, se consigue minimizar el error dando por hecho que el diseño de la prueba se ha realizado correctamente.

3.2.1. Datos obtenidos

Cuando se desea diseñar un test de escucha hay que tener en cuenta: la cantidad de sonidos a evaluar, la duración de estos y los parámetros subjetivos a juzgar.

Dependiendo de a quién desee realizar la prueba de escucha, se podrá poner más o menos audios, por ejemplo en [7] donde los jueces eran niños, los audios tenían que ser cortos y en un número pequeño, porque si no pierden el interés por hacer bien la prueba y resultan datos no válidos.

El número máximo de parejas en un test está directamente relacionado con el número de sonidos, ya que se rige por la siguiente fórmula:

$$N_{comp} = N_s * (N_s - 1); \qquad N_s \equiv Número\ de\ sonidos\ a\ comparar \tag{1}$$

En el número de comparaciones máxima se incluyen todas las permutaciones entre los diferentes sonidos sin coincidir el sonido consigo mismo en un par. Por ejemplo, si en un test de 3 sonidos eligiéramos todas las combinaciones (6 máximo) se tendrían las parejas mostradas en la tabla 2.

	1	2	3
1		x	x
2	x		x
3	x	x	

Combinaciones
1 vs. 2
1 vs. 3
2 vs. 1
2 vs. 3
3 vs. 1
3 vs. 2

Tabla 2: Combinaciones de pares de sonidos de ejemplo

Los números representan al *Sonido 1*, *Sonido 2* y al *Sonido 3*. Estos sólo se pueden combinar entre sí y no consigo mismos, por ello la diagonal principal de la matriz está en color gris.

Las combinaciones mostradas en la tabla 2 son en orden. Con esta cantidad de pares ya se podría definir un test, solo que no sería un buen diseño ya que algunos sonidos se reproducen en la misma posición dos veces seguidas (1 vs.2 y el siguiente 1 vs.3).

Es más óptimo mezclar los pares los cuales se vayan a repetir de un par a otro seguido, siendo la repetición de un sonido en la posición A (izquierda) o en la posición B (derecha). Si se deseara evaluar los tres sonidos pero sin añadir las 6 combinaciones, se podría, siempre y cuando existiera una o dos combinaciones permutadas, es decir que estuviera la *1 vs. 2* y la *2 vs. 1* como mínimo (como se explica en el siguiente apartado). Los diferentes valores que se obtienen están relacionados con los jueces, las características subjetivas de los sonidos y sobre los sonidos en sí mismos.

3.2.1.1. *Datos obtenidos sobre los jueces*

Para decidir si un juez en la prueba es válido o no se utiliza el parámetro de repetitividad.

1. **Repetitividad:** Representa las veces que un mismo juez ha mantenido la misma opinión en las ocasiones donde se le ha planteado la misma comparación permutada. Se indica como un porcentaje. Si se tuviera el caso del ejemplo de la tabla 2, se podría tener 0%, 33%, 66% o 100% de repetitividad, ya que hay tres comparaciones que aparecen permutadas en el mismo test. En este caso, para decidir si un usuario es válido o no, lo adecuado sería elegir como umbral un valor del 66% de repetitividad. Con ello aseguramos que puede haberse equivocado en una sola permutación pero que las demás las elige a conciencia. Dicho porcentaje de repetitividad se calcula mediante una simple fórmula después de recolectar los datos de elección:

$$Repetitividad\ (\%) = \frac{Nº\ de\ pares\ con\ la\ misma\ selección\ en\ ambos\ pares\ permutados}{Nº\ total\ de\ pares} * 100 \tag{2}$$

Para conseguir mayor resolución de datos con respecto a los jueces, se debe tener en cuenta no solo si eligen en una misma comparación el mismo audio, sino si sus elecciones tienen coherencia con respecto a todos los audios de la prueba. El parámetro para evaluar la coherencia es denominado consistencia.

2. **Consistencia:** Indica el grado de coherencia de las opiniones de cada juez en porcentaje, siendo este un indicador de si mantiene su criterio a la hora de evaluar o por el contrario cambia. Para calcularlo se utiliza la estadística de Kendall como en [6] , la cual se basa en la idea de triadas circulares para inconsistencia y no circulares para que sea consistente.

La idea es que teniendo los sonidos 1, 2 y 3, se es consistente si 1 es preferido a 2, 2 es preferido a 3 y 1 es preferido frente a 3. Por ello es importante que uno de los parámetros subjetivos sea la preferencia, éste por tanto en *Sound Challenge* será el parámetro principal que siempre aparecerá.

En la figura 2 aparece un ejemplo de test donde se tienen tres audios a evaluar pero solo se han elegido en vez de 6 pares, 4. Es una representación esquemática de una prueba de escucha. En concreto este ejemplo muestra las elecciones de un juez 100% repetitivo y 100% consistente. Se puede comprobar que en la primera comparación selecciona el sonido 2 frente al 1, en la siguiente comparación prefiere el 3 frente al 2 corroborando por tanto en la tercera comparación prefiere el 3 frente al 1, esto demuestra la consistencia. El último par es el que confirma que es 100% repetitivo ya que es la misma comparación que el primer par pero en diferente orden.

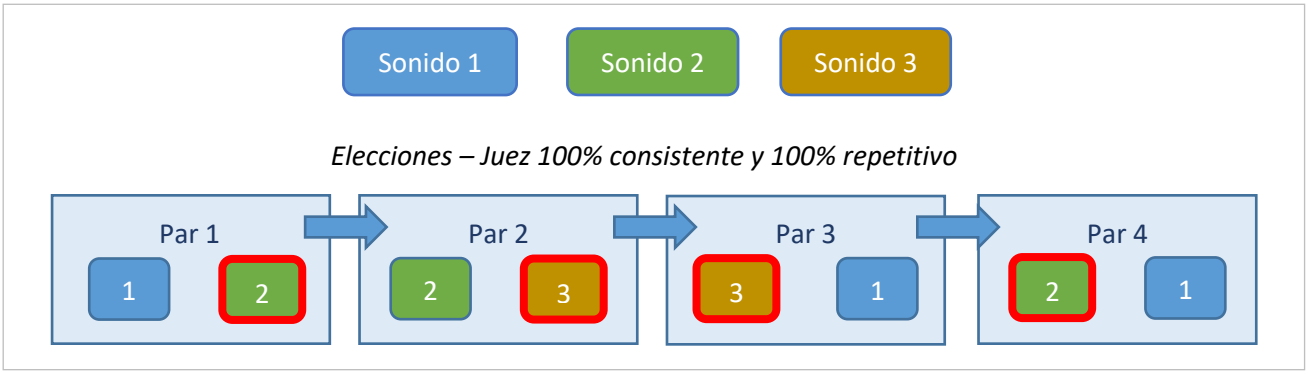


Fig. 2: Representación de la repetitividad y la consistencia en una prueba de escucha

Por otro lado, se es inconsistente si 1 es preferido a 2, 2 es preferido a 3 pero 3 es preferido a 1. Puede ocurrir que no se pueda evaluar la consistencia de manera fiable, por ejemplo que 1 sea preferido a 2, 3 es preferido a 2 y por tanto en la comparación de 3 a 1 no importa cual se elija, esta triada no sería válida para la consistencia. Por ello, para *Sound Challenge*, se optó que el valor umbral el cual decidiera qué jueces valdrían sería solo la repetitividad. Ambos parámetros se calculan mediante una matriz llamada matriz de preferencias, donde se indica si se ha preferido el sonido A o el sonido B en cada combinación de pares [7].

3.2.1.2. *Datos obtenidos sobre las características de los sonidos*

Para averiguar si la opinión de los usuarios es relevante o no con respecto a una característica subjetiva de los sonidos, se realiza con la concordancia.

- **Concordancia:** Es el nivel de acuerdo entre los diferentes jueces de un mismo test sobre las características evaluadas. Se cuantifica con un coeficiente que puede tener valores entre 0 y 1, siendo el valor mínimo que no existe acuerdo y viceversa [7]. Se calcula de la siguiente manera:

Suponiendo m jueces que tienen que evaluar n pares de comparación. El coeficiente de concordancia entre los veredictos de los m jueces es dado por (3) donde Σ es el sumatorio del número de veces que los jueces están de acuerdo sobre los pares.

$$C = \frac{8\Sigma}{m(m-1) * 2 * número_{comparaciones}} - 1 \qquad (3)$$

El resultado obtenido se puede interpretar en que si no existe acuerdo, puede no ser relevante dicha característica ya que las opiniones son muy dispares o quizás hay que realizar un sesgo mayor de los jueces, por ejemplo, diferenciándolos por rangos de edad para averiguar si finalmente están de acuerdo o no.

3.2.1.3. *Datos obtenidos sobre los sonidos*

El objetivo principal de este tipo de pruebas es conseguir información de los sonidos que la componen. De ellos se puede conocer el grado de preferencia o la opinión que tiene el jurado de ellos para cada una de las características. Estas calificaciones de los sonidos son tanto cuantitativas como cualitativas ya que se pueden obtener 'notas' de cada sonido para cada una de las características, lo que permitiría clasificar los sonidos. Dichas valoraciones están basadas en el modelo de *Bradley-Terry* [7] el cual se fundamenta en que la valoración subjetiva de una cierta característica es una variable aleatoria con una distribución de probabilidad determinada. Se obtienen dos parámetros en relación a los sonidos:

1. **Probabilidad:** Se calcula a partir de la matriz de preferencias antes comentada utilizando el modelo de *Bradley-Terry*. Cada sonido se considera como una variable independiente aportando así, mediante un cálculo iterativo, un valor entre 0 y 1. Cuanto más cercano a 1 más veces lo ha elegido los jueces en relación a la preferencia. Para que sea más representativo, en *Sound Challenge* se muestra en porcentaje.

2. **Valores de mérito (VM)**: Representan las valoraciones de las características para cada sonido. La suma de todas las valoraciones debe ser igual a 0 ya que algunas tienen valores negativos. La interpretación sería que por ejemplo la característica 'Molestia' en el sonido 1 tiene valor -6 y en cambio para la 'Preferencia' tiene un valor 6, esto es que los jueces han determinado que el sonido 1 es de los menos molestos. Por otro lado el VM no tiene un valor máximo y mínimo, pero sí permite distinguir si esa característica es importante o no, de forma que cuanto mayor sea su rango dinámico, más significativo es.

En el capítulo 5 de este mismo documento se muestran con un ejemplo todos los datos obtenidos mediante la aplicación *Sound Challenge*.

Capítulo 4. Introducción a la programación web

Se detalla de manera breve pero concisa las herramientas empleadas para la creación de la aplicación web. Empezando por una breve explicación de la programación web destacando las características más relevantes de HTML5, CSS3 y JavaScript, continuando con la explicación del *framework* Django y las librerías utilizadas, después está la explicación del *framework Bootstrap* para crear diseño web responsivo y finalizando con la definición del sistema de control de versiones utilizado. De esta manera se explican los conceptos más relevantes para comprender el procedimiento de desarrollo y la dificultad que conlleva un desarrollo web.

Capítulo 4. Introducción a la programación web

En todo proceso de desarrollo el primer paso es determinar qué necesidades y requisitos se tienen realizando un análisis previo para elegir qué lenguaje y herramientas son los óptimos. Dependiendo del objetivo de diseño que se quiera abarcar, se utiliza un lenguaje u otro, además también influye la experiencia que se haya tenido sobre la utilización de los mismos si es un proyecto el cual no confluyen más desarrolladores como es el caso. Si no se conoce a priori el entorno en el que se va a trabajar y las posibilidades que el lenguaje tiene, un programa simple puede hacerse cuesta arriba.

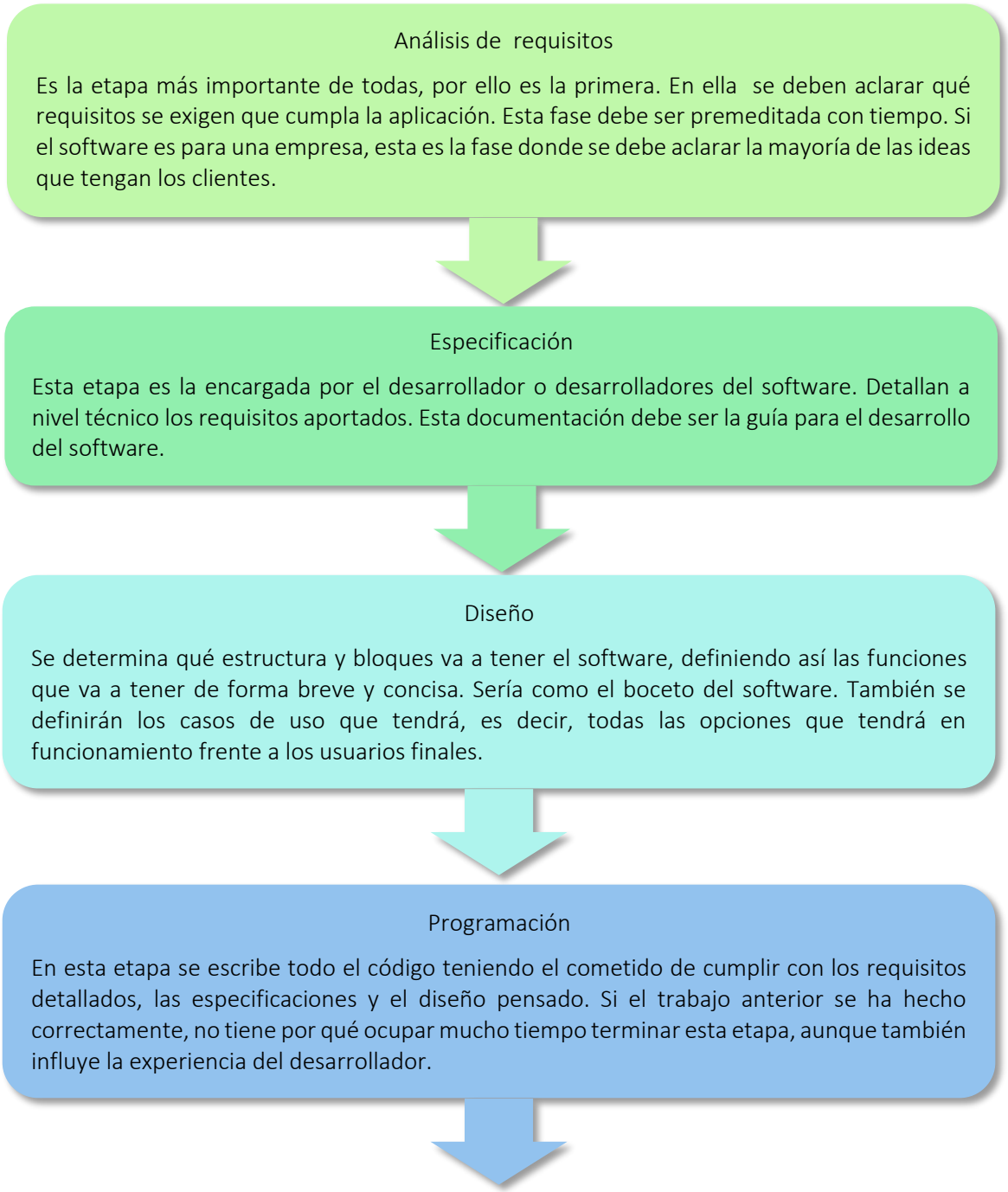
Al comienzo del presente trabajo, los conocimientos que se tenían eran básicos sobre HTML5, JavaScript y CSS3, por lo que hubo ciertas dificultades durante el desarrollo de la aplicación por la inexperiencia en el campo. Pero para suplir uno de los escalones más complicados, durante el análisis de los requisitos de la aplicación, se eligió el *framework* Django para facilitar la creación de la aplicación, ya que permite con conocimientos básicos, crear un sistema completo, sin preocuparse de conocer más lenguajes que los anteriormente nombrados, solamente hay que conocer programación básica de Python, la cual se tenía. La creación de tablas, administración e interacción de las mismas se realiza mediante librerías e instrucciones en dicho lenguaje.

Para paliar la inexperiencia, lo primordial fue tener claro qué se iba a necesitar en el programa y cómo realizarlo. Uno de los prerrequisitos en este trabajo en concreto, fue el estudio de la estructura y las funcionalidades que permite Django, lo cual se explica más adelante, pero primero hay que destacar que existen diferentes etapas por la que todo desarrollo de software pasa desde que se tiene la idea hasta que se realiza además dependiendo de una metodología u otra se definen diferentes ciclos de vida que tiene el software [8].

En este capítulo primero se explican las diferentes etapas que posee un desarrollo de software explicando en él la metodología empleada en *Sound Challenge* por su adecuación a los requisitos del proyecto. En el siguiente punto se habla del concepto de Modelo-Vista-Controlador el cual lo pone en práctica el *framework* Django y que se ha aplicado igualmente en el desarrollo de las diferentes partes de la aplicación web. Para enmarcar qué tipo de programación se ha utilizado, se explica a grandes rasgos los lenguajes de programación HTML5, JavaScript y CSS3 mostrando así su potencial y la relación que existe entre ellos. Después se explica Django de forma esquemática y no muy complicada destacando los elementos utilizados ya que es uno de los *frameworks* más completos que hoy en día hay. Dentro de esta parte se indica que la base de datos relacional que implementa el *framework* es un módulo de Python llamada SQLite3. Después se comenta qué es una aplicación web adaptativa la cual en este caso para *Sound Challenge* se ha optimizado con el *framework* Bootstrap. Y finalmente se explica por qué es necesario un sistema de control de versiones y cual se ha empleado.

4.1. Etapas de desarrollo y ciclo de vida de *Sound Challenge*

Realizar un proyecto de software no es algo trivial, se deben seguir diferentes pautas para llegar a desarrollar un trabajo sólido y con el menor número de errores posibles para que al final del mismo se pueda mejorar sin muchas complicaciones. Las etapas que ha seguido el desarrollo de *Sound Challenge* y que debería llevar cualquier desarrollo son las siguientes:



Prueba

Es la etapa donde se realizan diferentes tests para descubrir errores si los hubiera y todas las funcionalidades que tiene. En esta etapa se decide si el programa está listo para salir a producción o se deben hacer modificaciones. Las pruebas las debe realizar alguien distinto al desarrollador para que sean más fiables.

Mantenimiento

Es la última etapa. Después de enviar el software a producción es importante que deba ser revisado cada cierto tiempo para asegurar la viabilidad del mismo en el mercado, ya que puede acabar desactualizado con respecto a las plataformas web que existan o pueden aparecer errores los cuales hay que subsanar. Por otro lado esta etapa también sirve para la posterior actualización y mejora del software.

Las etapas explicadas son fases generales que todo software debe tener, para cumplir dichas fases existen diferentes metodologías de desarrollo las cuales se caracterizan por el tipo de gestión y capacidad de cambios. Ayuda a controlar las actividades del proyecto de inicio a fin. En concreto *Sound Challenge* ha tenido un ciclo de vida en V (figura 3) el cual su característica más destacable es que cada paso está pensado para asegurar la calidad del software.

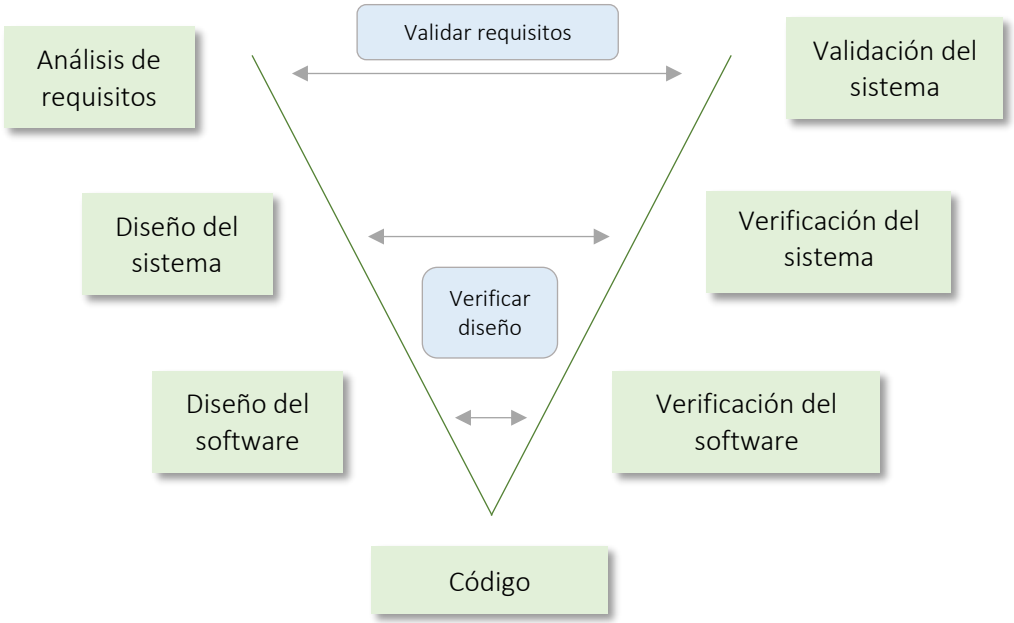


Fig. 3: Ciclo de vida de software en V

Este método a la par de simple resulta mucho más eficiente que otros, ya que en cada etapa del proyecto se verifica que se están tomando las decisiones correctas y esto previene errores futuros. El único inconveniente es que es un método rígido, todas las partes están claramente definidas, por tanto este tipo de metodología es buena para proyector pequeños. Por ello se hace óptimo para este trabajo.

4.2. *Concepto Modelo-Vista-Controlador (MVC)*

Como en toda ingeniería, lo importante es conseguir un trabajo o proyecto que sea eficiente tanto en funcionalidad como en la implementación. En la ingeniería de software una de las tareas más importantes en el desarrollo es conseguir crear funciones lo más flexibles posibles. En lenguajes de programación orientados a objetos (como JavaScript y Python) es importante estructurar el programa en el máximo número de bloques independientes que se puedan, ya que la idea fundamental es poder reutilizarlos las veces que se necesiten. Este tipo de filosofía es la que se define cuando se habla de Modelo – Vista – Controlador (MVC) en programación web. Cada uno de ellos representa una parte fundamental que tiene el software. Esta metodología surgió gracias al aumento de complejidad que añadía implementar una interfaz de usuario. Se necesitaba crear un software más robusto donde se facilitara el mantenimiento, la reutilización del código y la separación de conceptos [9].

El objetivo de ello es separar el código en tres capas diferentes dependiendo de las responsabilidades y funciones que tenga. La identificación de dichas capas es primordial para hacer un código limpio [10]:

- **Modelo**: Es la capa donde se trabaja con los datos. Su cometido es procesar, administrar, servir y modificar los datos. Principalmente es la gestión de la base de datos y su modificación.
- **Vista**: Como su nombre indica, es la parte que se 've' de la aplicación web. Es el código HTML, CSS y parte del JavaScript, que muestra la manera en que ha sido pensada la información y los elementos para interactuar con el modelo. Es la capa donde está la visualización de la interfaz de usuario.
- **Controlador**: Es la etapa del software donde está todo el código que controla las acciones que van a realizar los elementos que están en la vista. Es el enlace entre la vista y el modelo. Su responsabilidad no es la de sacar ninguna salida, solo es el medio de comunicación para que se ejecuten las funcionalidades de la aplicación.

Para comprender más claramente cómo funciona cada etapa, a continuación se define un diagrama de flujo donde aparece de forma esquemática las interacciones entre cada etapa teniendo como ejemplo que el usuario hace la acción de registrarse en la aplicación web:

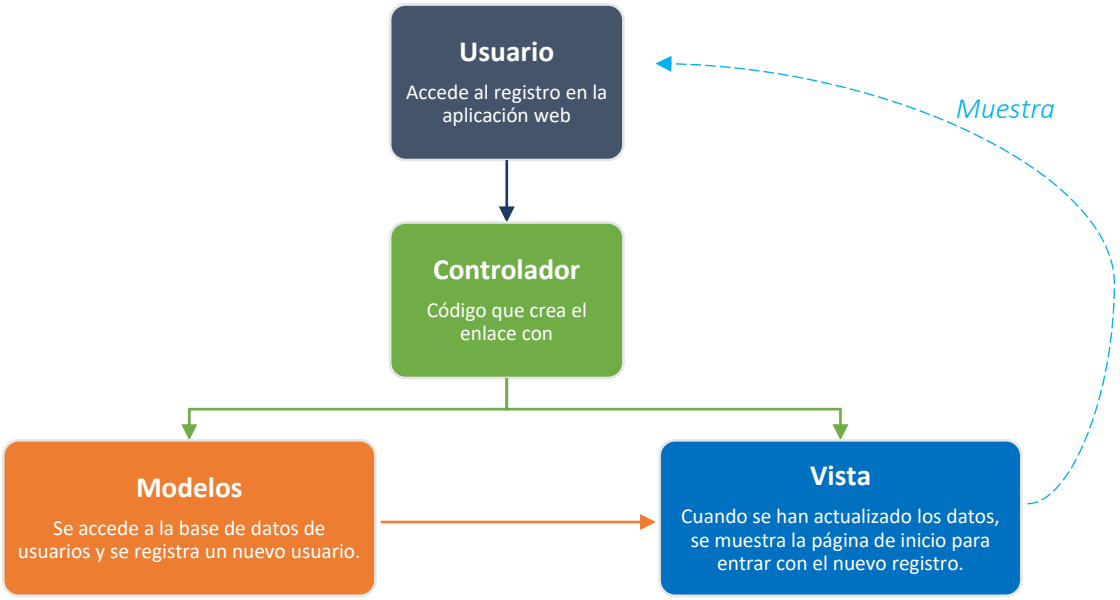


Fig. 4: Diagrama de flujo del método MVC

4.3. *Desarrollo web*

Hasta ahora se han definido diferentes conceptos generales correspondientes al desarrollo de software, pero en concreto este trabajo consta del desarrollo de una aplicación web. Las herramientas o lenguajes de programación que se utilizan para desarrollar una aplicación o página web, difieren en relación a si se está programando la parte más visual (vistas) o la parte de administración de bases de datos y control (modelos y controlador). Normalmente en un proyecto grande participan diferentes desarrolladores, organizados para que unos realicen dichas partes (cliente o servidor), se denominan desarrolladores *front-end* o *back-end* respectivamente.

- Desarrollador *front-end*: Se encargan de desarrollar la aplicación o página web para que sea cómoda de usar a la par que eficiente. Se correspondería con desarrollar la etapa de la vista y parte de la etapa del controlador definido en el método de MVC. Se denomina que trabaja en el lado del cliente, y debe implementar los lenguajes de HTML para la estructuración de la web, CSS para dar estilo a la misma y JavaScript para añadir dinamismo a la web. En ningún momento deben crear código que almacene datos en el cliente, de eso se encargan en el *back-end.*

- Desarrollador *back-end*: Su tarea es crear el código necesario para administrar los datos e información que necesita la web. Manejan la parte del servidor, gestionándolo. Este tipo de desarrollador trabaja en la etapa de modelos y parte de la del controlador del método MVC. Hay diversidad de lenguajes en los cuales puede trabajar, en concreto para el desarrollo de *Sound Challenge*, se ha programado en Python ya que se ha utilizado el *framework* Django, utilizando por tanto la base de datos facilitada por este, SQLite3.

Ambos desarrolladores se necesitan el uno del otro para que la web finalmente funcione, pero existen casos sobre todo cuando se tratan de proyectos más pequeños o empresas pequeñas, dónde un solo desarrollador implementa ambas partes además de definir todas las fases del desarrollo web, se denomina *full-stack* [11]. Su deber es conocer todos los lenguajes de programación necesarios, aunque a veces sabe más de una de las partes.

En concreto este proyecto ha sido desarrollado de manera *full-stack* lo cual ha sido un gran reto para conseguir realizar correctamente todas las fases. Afortunadamente, al utilizar Django, se elimina la necesidad de programar en un lenguaje de programación de servidores como PHP, ya que para realizar un proyecto pequeño es suficiente con conocer Python. En este caso, ya que la gestión de las tablas de la base de datos se podía realizar mediante el *framework* y se tenía mayor experiencia con la programación del *front-end*, la mayor parte del código fue de HTLM5, CSS3 y JavaScript que a continuación se describe cada uno de ellos de forma breve.

4.3.1. HTML5

HTML, sigla en inglés de *HyperText Markup Language* es el lenguaje de programación de marcado para crear páginas web. Se basa en un estándar el cual con diferentes etiquetas nombradas se define el contenido de la web como texto, imágenes, videos, audios, etc. Fue estandarizado por *World Wide Web Consortium* (W3C) quienes también estandarizan prácticamente todos los lenguajes relacionados con la web [12].

La estructura de código es muy sencilla, está formada por una etiqueta de apertura, el contenido y una etiqueta de cierre como se puede observar en la figura 5. Dependiendo del tipo de elemento puede existir contenido escrito o más etiquetas dentro de unas y otras anidadas, e incluso hay algunas que no necesitan cerrarse para definir el elemento.

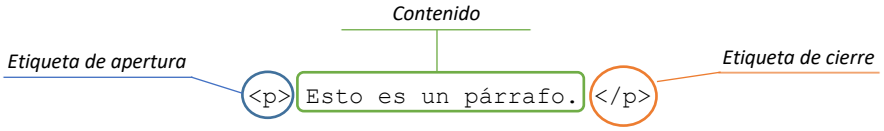


Fig. 5: Estructura básica elemento HTML - párrafo

Los elementos HTML poseen los llamados atributos los cuales determinan algo en concreto del elemento, configuran al elemento. Por ejemplo, en un elemento que representa un botón, se puede deshabilitar simplemente poniendo *disabled* como atributo:

```
<button disabled>Púlsame</button>
```

Como se ha visto, cada etiqueta hace referencia a un elemento o parte de la página web. En toda página web existen unas etiquetas comunes las cuales son suficientes para que se considere una web. En la figura 6 se define la estructura básica que toda página web debe tener.

```
<!DOCTYPE html>
<html lang="en">
<head>
       <meta charset="UTF-8">
       <title>Title</title>
</head>
<body>
<!-- Esto es un comentario -->
</body>
</html>
```

Fig. 6: Estructura básica HTML

Los elementos que aparecen tienen este significado:

`<!DOCTYPE html>` - Instrucción para el navegador acerca del tipo de documento; no es propiamente una etiqueta. Esta indicación es propia de HTML5.

`<html lang='en'></html>` - Etiqueta que define al documento HTML teniendo como atributo el lenguaje que se va a emplear dentro de este, el cual es este caso es el inglés.

`<head></head>` - Elemento de cabecera dónde aparecerán los metadatos necesarios para una página web, como el elemento `<meta charset="UTF-8">` que define el tipo de codificación de caracteres que se va a utilizar, en este caso es la UTF-8. También se introducen dentro del *head* elementos como el título que aparecerá en la pestaña de la aplicación, se define con `<title></title>`.

`<body></body>` - Es la parte donde irán la mayoría de elementos los cuales serán los que estructuren la página web y añadan funcionalidades.

Como se puede observar en la figura 6, el elemento *head* y *body* están dentro del elemento *html* y a su vez los elementos *meta* y *title* están dentro del elemento *head*. La jerarquía es uno de los fundamentos del HTML, se define como parentesco [12]. El parentesco significa que el elemento *html* es padre de *head* y *body*, donde *head* además de ser hijo de *html* es padre de *meta* (1er hijo) y *title* (2º hijo). Esto define qué elementos están dentro de quien para definir tanto su posición, dimensión, apariencia y funcionalidad. Uno de los elementos más comunes dentro del *body* para definir la estructura y dimensiones de la web es uno denominado *div*, el cual es uno de los elementos principales en la aplicación de *Sound Challenge*.

Un elemento *div* es un elemento tipo bloque, el cual puede contener cualquier otro tipo de elementos e ir anidándolos. Hay otro tipo de etiquetas que permiten estructurar la página web estos tienen predefinidas diferentes formas y apariencia, todas estas etiquetas se pueden consultar en [13]. Pero en *Sound Challenge* fue más adecuado utilizar los elementos *div* ya que permitían mayor versatilidad de formatos y mejor implementación del *framework* Bootstrap que más adelante se explica. Para definir los tamaños y posiciones de cada elemento se utiliza CSS.

Hay diferentes versiones de HTML, hoy en día la versión más actualizada es la HTML5 la cual se ha utilizado para desarrollar *Sound Challenge*. La diferencia con las demás versiones es que añaden más elementos y funcionalidades [12], los cambios más importantes son:

1. Multimedia nativa: Se puede reproducir audio y video en la misma página web de forma nativa, sin necesitar ningún *plug-in*. Son los elementos *audio* y *video.*
2. Superficie de dibujo genérica: Elemento *canvas* que permite dibujar gráficas, de esta forma no es necesario utilizar el Adobe Flash, solo hay que utilizar JavaScript para complementar al elemento.

4.3.2. CSS3

CSS, sigla en inglés de *Cascading Style Sheet*, significa Hoja de estilo en cascada, es un lenguaje de diseño gráfico para definir cómo se van a presentar los elementos de lenguaje de marcado como el HTML. Se denomina 'en cascada' porque el estilo se aplica dependiendo de ciertas características que dotan de preferencia a una instrucción u otra. Se creó CSS para separar el contenido de la web y el estilo de la misma, para así un mismo estilo fuera reutilizable por varios elementos, realizando capas y clases.

La separación del formato y el contenido hace posible que por ejemplo para diferentes tamaños de pantalla sea posible cambiar las fuentes, el tamaño, las formas, CSS permite que la visualización sea dinámica. Todas las posibles características modificables por CSS se encuentran explicadas en [14], a continuación se muestra unos pocos ejemplos de propiedades de estilo sobre los elementos HTML. Se define como ***propiedad: valor***.

PROPIEDAD	DESCRIPCIÓN
background-color	Define el color de fondo de un elemento.
height	Altura de un elemento
width	Anchura de un elemento
font-size	Tamaño de la fuente de un elemento

Tabla 3: Especificación de propiedades de estilos

Para aplicar las propiedades de estilo a los elementos que están en el archivo HTML, existen cuatro maneras de declaración estando ordenados por nivel de preferencia:

1. Estilo en *línea:* Se define en los elementos cuyo padre es el *body*. Se especifica como un atributo del elemento llamado *style* y dentro de este, entre comillas se definen las propiedades. Tiene preferencia entre los demás tipos de declaración. El ejemplo de a continuación se aplica un borde fino solido en negro al elemento *span* que tiene como contenido la palabra 'saludo'.

```
<body>
   <p> Párrafo de
      <span style="border:thin black solid">
         saludo
      </span>
   </p>
</body>
```

2. Estilo *embebido*: Se define de forma explícita en el *head* del archivo HTML. Se especifica con la etiqueta *style* definiendo su atributo de tipo (*type*) como texto de css (*text/css*). Esta definición del estilo afecta a todos los elementos que estén dentro de *body* que se hayan seleccionado. En el siguiente ejemplo de estilo se aplica color de fondo en gris y color de letra en blanco a todos los elementos *a* (*anchor*).

```
<head>
   <title>Ejemplo de estilo embebido</title>
   <meta charset="utf-8"/>
   <link rel="icon" href="favicon.gif" type="image/gif"/>
   <style type="text/css">
    a{
       background-color:grey;
       color:white
    }
   </style>
</head>
```

3. Estilo *externo*: Se importa un fichero de estilo (ej.: *estilo.css*) dónde se especifican las propiedades de la misma manera que en la declaración embebida, entre corchetes. Tiene la misma preferencia que el embebido. La importación se realiza con un elemento *link* siendo su padre el elemento *head*. A continuación se muestra la importación definiendo como atributo del elemento *link* que se trata de una hoja de estilos (*rel="stylesheet"* y *text="text/css"*), la dirección donde se encuentra el archivo se define con el atributo *href="estilo.css".* Si el archivo no se encuentra en el mismo directorio que el archivo HTML, hay que escribir toda la dirección completa. La hoja de estilos externa se muestra debajo del ejemplo.

```
<head>
   <title>Ejemplo de estilo externo</title>
   <meta charset="utf-8"/>
   <link rel="icon" href="favicon.gif" type="image/gif"/>
   <link rel="stylesheet" type="text/css" href="estilo.css"></link>
</head>
```

Archivo: estilo.css

```
a{
    background-color:grey;
    color:white
}
span{
    border:thin black solid;
}
```

4. Estilo *importado* de otra hoja de estilos: Dentro del archivo de *estilo.css* se pueden importar otros estilos. Para que se anide las diferentes especificaciones de propiedades se tiene que poner antes de las mismas especificaciones CSS las siguientes instrucciones:
    - Sentencia **import**: Importa otro fichero de estilo (puede ser recurrente)
    - Sentencia **charset**: especificación de codificación del fichero (UTF-8 por defecto si no está especificado).

Se importa de la misma manera que la declaración embebida, dentro del *head*. Se tendría por tanto dentro del archivo *estilo.css* las siguientes instrucciones:

Archivo: estilo.css

```
@charset "utf-8";
@import "estiloParrafo.css";
a{
   background-color:grey;
   color:white
}
span{
   border:thin black solid;
}
```

Archivo: estiloParrafo.css

```
p{
   color: green;
   width: 10%;
   height: 20%;
   position: absolute;
}
```

Hay infinidad de propiedades, algunas propias de los elementos y otras generales, además hay muchas formas de seleccionar los elementos que se desean añadir estilo, la explicación de la selección de elementos no entra dentro del propósito de este capítulo aunque se puede consultar en [14] y [15]. Pero hay un último detalle el cual es importante para entender cómo CSS define el tamaño de los elementos y su posición: el modelo de caja.

El modelo de caja define las partes que tiene un elemento HTML cualquiera. Todos los elementos tienen las propiedades para definir una caja, de esta manera se tratan como si fueran bloques. En la figura 7 se muestra de forma esquemática las diferentes partes que tiene todo elemento HTML.

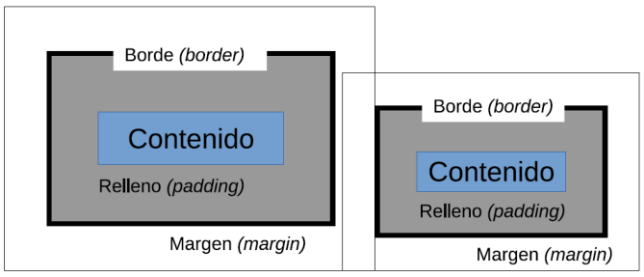


Fig. 7: Modelo de caja

Para cada parte de la caja existen muchas propiedades que las definen. Una característica importante a destacar es la diferencia que hay entre el relleno (*padding*) y el margen (*margin*). Dos elementos pueden solapar sus márgenes pero no su relleno, como se puede observar en la figura 7.

Por último añadir que igualmente como HTML5, CSS3 es la versión más actualizada de las hojas de estilo, en concreto la versión 3, la cual añade muchas más opciones de visualización y adaptación con todos los dispositivos (*smartphones, tablets…*) [16]. Aunque siguen habiendo

incompatibilidades de instrucciones para los diferente navegadores, por tanto se deben seguir utilizando distintas instrucciones concretas para habilitar algunas propiedades por ejemplo para Mozilla (*-moz*).

4.3.2.1. Polymer 2.0

Hoy en día hay varios *frameworks* y elementos ya creados con el fin de facilitar el diseño a los desarrolladores web. En *Sound Challenge* a parte de utilizar el *framework* Bootstrap, se implementó una serie de elementos ya creados obtenidos del proyecto Polymer. Es un proyecto de código abierto hecho por un equipo de desarrolladores *front-end* que son parte de la organización de Chrome de Google [17]. Este proyecto tiene como objetivo facilitar el diseño con elementos HTML propios los cuales tienen características muy representativas de Google y resultan atractivas.

La ventaja de este proyecto frente a otros, es que han implementado los elementos HTML con CSS y JavaScript puro. Esto permite que sea fácil la implementación en la mayoría de navegadores. Fue pensado para que la programación además de ser más ligera, fuera sencillo implementarlo para todos los formatos de pantalla que hoy existe.

Cuando se comenzó a desarrollar *Sound Challenge* se utilizaron gran variedad de elementos de Polymer, pero se prefirió utilizar el *framework* Bootstrap para que la aplicación fuera con un diseño adaptativo y ello conllevó a eliminar gran cantidad de elementos ya que fue más complicado juntar Bootstrap con Polymer, que simplemente utilizar Bootstrap el cual ofrecía más flexibilidad y rapidez dentro del *framework* Django. Además, la ventaja que introduce Polymer en cuestión de compatibilidad, también la tiene Bootstrap, es CSS y JavaScript puro. Finalmente se han mantenido algún botón e icono de Polymer, los cuales tienen etiquetas propias.

4.3.3. JavaScript

Como ya se ha visto, el código HTML sirve para crear la estructura y los elementos que componen la web, y CSS da la apariencia, tamaño y posición de dichos elementos. JavaScript (JS) surgió con la idea de dar comportamientos dinámicos a los elementos de la web siendo el mecanismo que permitía responder a las peticiones en la parte del cliente, en el navegador además de aportar más flexibilidad y opciones para dar más riqueza a las interfaces gráficas [18].

Es un lenguaje de programación interpretado el cual no necesita compilación para ser ejecutado, el navegador web interpreta instrucción a instrucción y las ejecuta. Es un lenguaje de programación orientado a objetos, basado en prototipos el cual los objetos no son creados mediante la instanciación de clases sino mediante la clonación de otros objetos o mediante la escritura de código por parte del desarrollador. De esta forma los objetos ya existentes pueden servir de prototipos para los que el desarrollador necesite crear. Los principales usos que se le puede dar son los siguientes:

- Añadir, cambiar y eliminar elementos HTML. Se pueden administrar cualquier tipo de elemento HTML siguiendo la jerarquía de parentesco.
- Cambiar atributos de los elementos HTML
- Establecer estilos, creando por tanto código CSS.
- Puede validar entrada de datos, por ejemplo en los formularios cuando no se ha rellenado algo o no está correctamente puesto. También puede enviarlos.

El código JavaScript son los llamados *scripts* los cuales se ejecutan de forma secuencial, instrucción a instrucción como se ha visto. Para añadir dichos *scripts* y que interaccione con los elementos HTML, existen dos maneras de hacerlo:

1. Importando un archivo *externo*: Igual que en CSS se puede importar un archivo con extensión *.js* (ej.: *acceso.js*) mediante el elemento *script* indicando con su atributo *src* dónde se encuentra el archivo JS en concreto. La importación es recomendable ponerla dentro del *head*:

```
<head>
   <title>Ejemplo de javascript externo</title>
   <meta charset="utf-8"/>
   <link rel="icon" href="favicon.gif" type="image/gif"/>
   <script src=”acceso.js"></script>
</head>
```

Archivo: acceso.js

```
// Función para cargar todas las señales de la BD
function get_signals(callback){
   //Consulta con AJAX
    var signals ='';
    $.ajax({
       url: "{% url 'subjT:get_data_signals' %}",
       type:'get',
       success: function (response) {
       //lo que haces si es exitoso
           var aux = response['ser_signals'];
           signals = JSON.parse(aux.replace(/&quot;/g, '"'));

           callback(signals)
          }
     });

  return signals;
}
```

2. Código implementado *internamente*: Se puede implementar exactamente el mismo código dentro del archivo HTML encerrándolo dentro de las estiquetas `<script type=”text/javascript”></script>`.

```
<html>
	<head>
		...
	</head>
	<body>
		...
		<script type=”text/javascript”>
			...
		// Función para cargar todas las señales de la BD
		function get_signals(callback){
		   //Consulta con AJAX
		    var signals ='';
		    $.ajax({
		       url: "{% url 'subjT:get_data_signals' %}",
		       type:'get',
		       success: function (response) {
		       //lo que haces si es exitoso
		           var aux = response['ser_signals'];
		           signals = JSON.parse(aux.replace(/&quot;/g, '"'));

		           callback(signals)
		          }
		     });

		  return signals;
		}
		</script>
	</body>
</html>
```

El ejemplo anterior es una de las funciones implementadas en la aplicación web *Sound Challenge*, la cual permite consultar la tabla de la base de datos donde están las señales que se van a utilizar en las pruebas de escucha. Se realiza la petición GET a la base de datos mediante una consulta AJAX (*Asynchronous JavaScript And XML*), es una técnica de desarrollo web que permite mediante JS acceso y ejecución asíncrona. Para que funcione en Django se tuvo que añadir un archivo JS para realizar la compatibilidad ya que surgían problemas con la verificación de peticiones que realiza Django. Si se quiere conocer las opciones que permite el lenguaje JS se puede consultar [19].

4.4. Django *en Pycharm*

Anteriormente se ha hecho un breve repaso sobre los lenguajes de programación utilizados principalmente para el desarrollo *front-end*, para este proyecto se ha utilizado el *framework* Django para terminar de definir la aplicación web, ya que no solo se define la parte *back-end* sino, con este *framework* se crea, gestiona y modifica la base de datos mediante Python, además de realizar el enlace apropiado para ejecutar los archivos HTML y servirlos.

Django es un *framework* web para crear aplicaciones web de forma gratuita ya que es código abierto. Está escrito en Python donde su finalidad es facilitar la creación de un servicio web completo de forma sencilla y siendo eficientes, por ello la frase por la que se rigen los desarrolladores de Django es: *No reinventes la rueda* [20]. Este marco de trabajo tiene definidas todas las herramientas necesarias para crear una aplicación web sin tener que ingeniárselas de nuevo, todo ello mediante módulos en Python, por ello la creación de cualquier aplicación web es mucho más rápida que programar todo desde cero, tanto para desarrolladores expertos como no expertos.

Para llegar a utilizarlo hay que instalarlo mediante Python tanto en Linux como en Windows, en concreto para el desarrollo de *Sound Challenge* en todo momento se ha utilizado Linux Ubuntu como sistema operativo. Todos los requisitos y comandos para poner en marcha el *framework* se puede consultar aquí [20]. Lo importante en este punto de la memoria es destacar la jerarquía que tienen los diferentes archivos dentro de una aplicación y cómo se comunican en ella. La jerarquía que se muestra en la figura 8 es la que permite visualizar el entorno de desarrollo que se ha ido utilizando a lo largo de este proyecto, el Pycharm 2017. La estructura sería la siguiente:

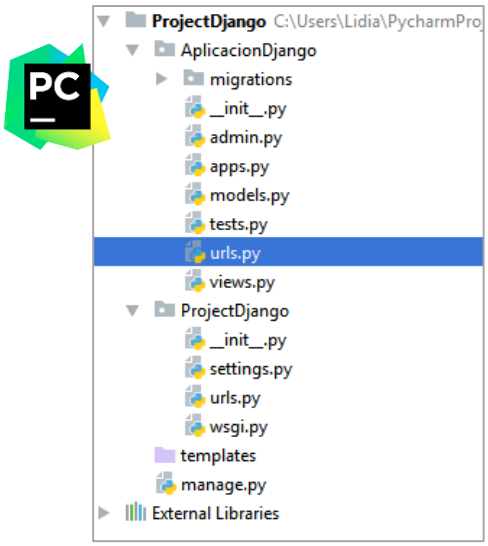


Fig. 8: Estructura jerarquíca de directorios de un proyecto y aplicación de Django en Pycharm

Elementos del proyecto relevantes y que se modificarán:

- *settings.py*: Archivo donde se especifican características de configuración y donde se importa la aplicación para que pueda funcionar.
- *urls.py*: Es donde se especifican las URLs para definir las aplicaciones. En un mismo proyecto se pueden tener varias aplicaciones a la vez.

Elementos de la aplicación relevantes:

- *urls.py*: Se debe crear a mano. Se especifican las URLs de las vistas.
- *apps.py*: Se define la aplicación con un nombre corto.
- *models.py*: Se especifican los modelos de la aplicación, esto es, las tablas que se crearán para guardar los datos que se quieran. En este archivo se define los **modelos del MVC.**
- *tests.py*: Django provee un sistema de depuración muy sofisticado, el cual se define en este archivo. Se programan situaciones para probar el código en concreto.
- *views.py*: Es uno de los archivos más importantes, en él se especifica el código necesario para acceder a las vistas y a los modelos. Se trata del **controlador del MVC.**
- *admin.py*: Se especifica qué elementos de los modelos se mostrarán en el administrador de Django.

Las **vistas del MVC** se especifican creando una nueva carpeta denominada *templates* en la raíz: *../ProjectDjango/AplicacionDjango*. Ahí se especificarán tanto los archivos HTML como los elementos estáticos de CSS y JavaScript.

Está basado en la metodología de Modelo – Vista – Controlador (MVC) y lo aplica de manera explícita. Como se ha visto antes, cuando se crea una nueva aplicación por defecto se crea una jerarquía de carpetas dentro del proyecto donde están tanto las vistas, los modelos y el controlador de la misma. Los elementos por defecto más relevantes son los siguientes:

1. Motor de base de datos relacional con el módulo SQLite3. Puede ser sustituible por otra base de datos.
2. Sistema administrador de Django. Permite de forma muy intuitiva manejar las tablas guardadas en la base de datos. El ejemplo que viene a continuación son las tres tablas implementadas para la aplicación de *Sound Challenge*, mostrando cómo se ve la tabla de las señales. Se accede poniendo en la URL: *dirección_servidor/admin*. En este caso se estaba ejecutando de forma local en el puerto 8000, por eso aparece *localhost:8000*.

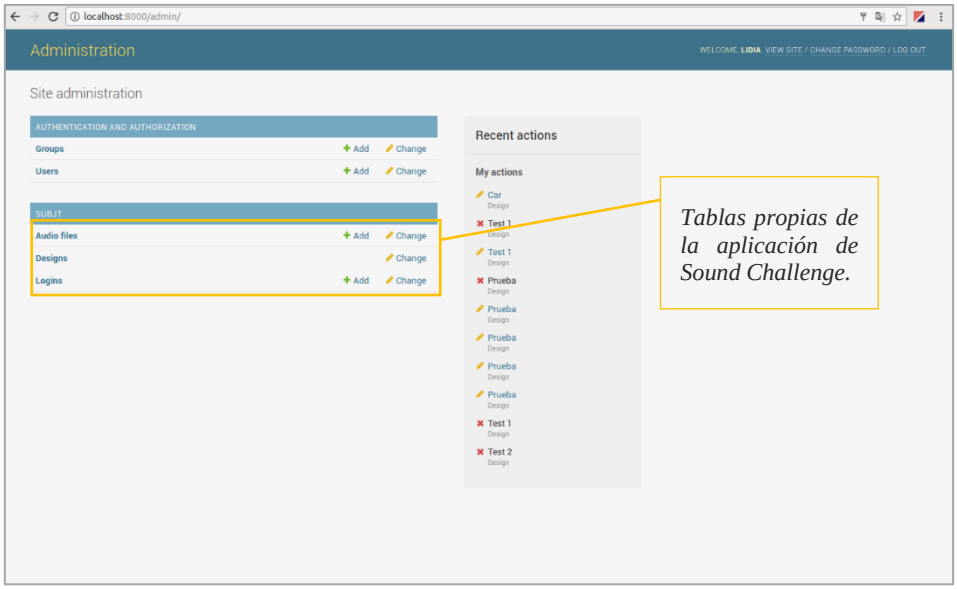


Fig. 9: Administración de Django con la aplicación de Sound Challenge

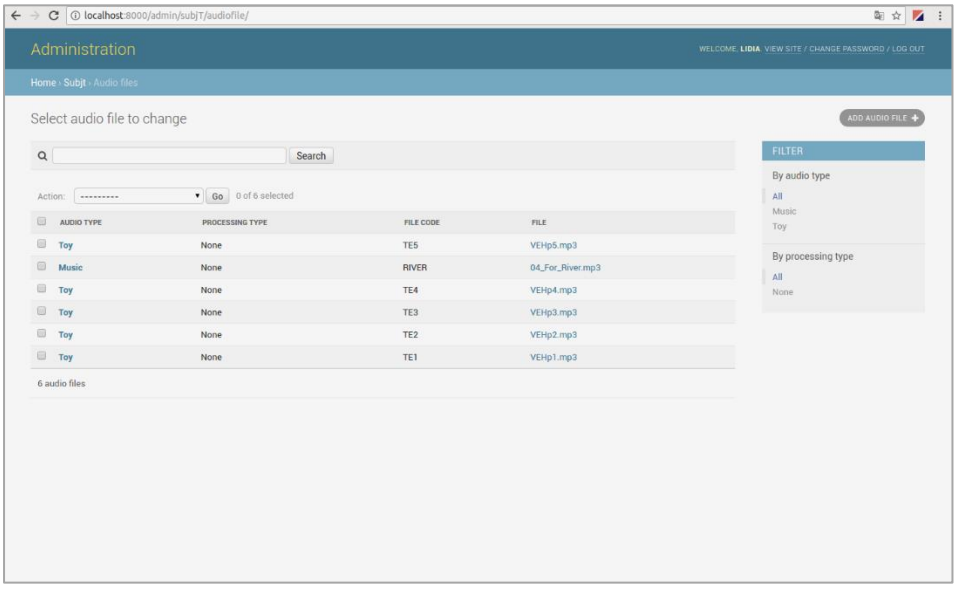


Fig. 10: Modelo de archivos de audio en Django

Dependiendo de las opciones que se describan dentro del archivo de administrador, se muestran unas u otras características. En la descripción de la aplicación en el capítulo 5, se detalla

en más profundidad qué campos tienen las tablas las cuales se denominan 'modelos' dentro de Django.

3. Archivo denominado vistas (*views.py*) que administran las vistas o *templates* (archivos HTML y demás) propias de la aplicación, este sería el controlador dentro del modelo MVC.
4. Archivo denominado *urls.py* que define las URL a donde debe dirigirse la aplicación para enlazar con una vista en concreto.

Finalmente para ejecutar una aplicación se hace con el siguiente comando o con la opción de ejecutar de Pycharm si se utiliza:

../ ProjectDjango/python manage.py runserver 'nombre_servidor:puerto_servidor'

Como se ha comprobado, en Django se tiene una estructura la cual tiene los datos enlazados de HTML, JS, CSS y base de datos mediante Python. Resulta ser un *framework* bastante seguro ya que tanto en verificación de usuarios como envío de formularios, todo lo hace de forma interna además añadiendo rutinas de excepciones. Algunas de las librerías o elementos concretos para *Sound Challenge* sobre Django se explican a lo largo del capítulo 5, pero igualmente, gracias al gran repositorio que ofrecen los desarrolladores del *framework*, en [20] está toda la información necesaria para saber qué módulos de python se implementan. Incluso añaden un tutorial sobre cómo realizar una página de encuestas y así comprender el funcionamiento básico de la estructura y organización que tiene.

El motor de base de datos implementada por defecto en Django es el módulo SQLite3. Esta es un paquete de software público que proporciona un sistema de gestión de bases de datos relacional. Un motor de base de datos como éste, además de administrar los registros, permite procesar consultas complejas combinando datos de varias tablas para implementar dichas relaciones. En Django las consultas se realizan en las *views.py*.

4.5. Bootstrap

Para conseguir tener una aplicación la cual se adapte a cualquier pantalla o dispositivo, hay que pensar en modificar tanto el tamaño de los elementos así como la posición y si estos se sustituyen por otros más compactos o no. Este cambio adaptativo se puede realizar desde cero con CSS, HTML y JS, pero sería tedioso, hoy en día existen muchos *frameworks* los cuales añaden diferentes librerías con código CSS y JS que permiten realizar un diseño web adaptativo sin mucho esfuerzo.

En concreto para este trabajo se eligió Bootstrap como *framework* de desarrollo *front-end* además de ser uno de los más populares. Se eligió porque utiliza código CSS típico y la implementación de los elementos resulta intuitiva [21]. Otra de las razones por la cual se decidió utilizar, fue porque se puede añadir como librería directamente a Django, funcionando como una aplicación más.

La filosofía que se rige Bootstrap es simple, todo se organiza por filas y columnas definidas con el elemento *div* dentro de otro *div* definido como *contenedor* (clase *.container*). Se rige por un sistema de rejilla donde cada fila puede tener 12 columnas en total. Dependiendo de cómo se quiere que se muestren los elementos y en qué dispositivo, se identifican a las filas y columnas con una clase en concreto. Dicha organización en rejilla obtenida de [21] define los tamaños y comportamientos de las columnas con prefijos de clases de CSS dependiendo del tamaño de dispositivo (tabla 4).

	Dispositivos extra pequeños *Móviles (<768px)*	Dispositivos pequeños *Tabletas (≥768px)*	Dispositivos medianos *Monitor de PC (≥992px)*	Dispositivos grandes *Monitor de PC (≥1200px)*
Comportamiento de la rejilla	Horizontal en todo momento	Colapsado al comenzar, se pone en horizontal en los puntos de ruptura entre columnas.		
Ancho del contenedor del *div*	Ninguno (auto)	750px	970px	1170px
Prefijo de la clase	`.col-xs-`	`.col-sm-`	`.col-md-`	`.col-lg-`
# de columnas	12			
Ancho de columna	Auto	~62px	~81px	~97px

Tabla 4: Opciones de rejilla de Bootstrap

También en [21] se muestran varios ejemplos como tutorial para implementar la rejilla que más convenga a la aplicación. Por ejemplo en la figura 11 se muestra la plantilla que se tomó para realizar la página principal de ingreso en la aplicación *Sound Challenge* (*index.html*).

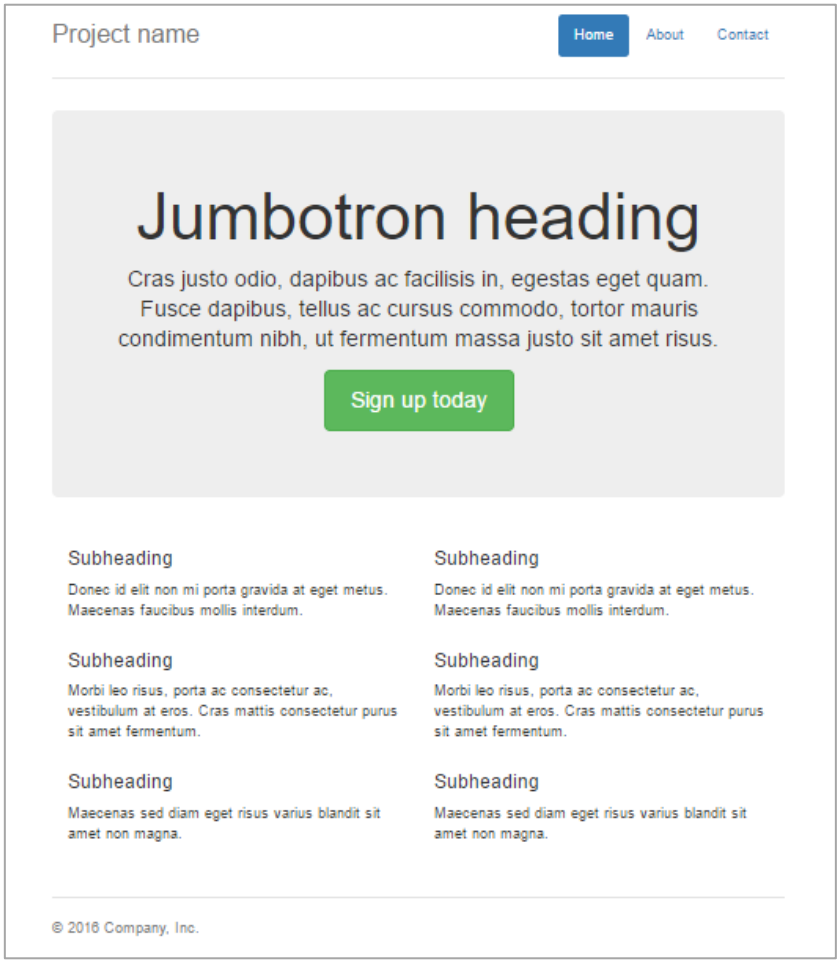


Fig. 11: Plantilla Jumbotron de Bootstrap

El código que hay detrás dentro del *body* del ejemplo es el siguiente:

```
<body>
  <div class="container">
    <div class="header clearfix">
      <nav>
        <ul class="nav nav-pills pull-right">
          <li role="presentation" class="active"><a href="#">Home</a></li>
          <li role="presentation"><a href="#">About</a></li>
          <li role="presentation"><a href="#">Contact</a></li>
        </ul>
      </nav>
      <h3 class="text-muted">Project name</h3>
    </div>

    <div class="jumbotron">
      <h1>Jumbotron heading</h1>
       <p class="lead">Cras justo odio, dapibus ac facilisis in, egestas eget
       quam. Fusce dapibus, tellus ac cursus commodo, tortor mauris
       condimentum nibh, ut fermentum massa justo sit amet risus.</p>
       <p><a class="btn btn-lg btn-success" href="#" role="button">Sign up
       today</a></p>
    </div>

    <div class="row marketing">
      <div class="col-lg-6">
        <h4>Subheading</h4>
        <p>Donec id elit non mi porta gravida at eget metus. Maecenas faucibus
         mollis interdum.</p>

        <h4>Subheading</h4>
        <p>Morbi leo risus, porta ac consectetur ac, vestibulum at eros. Cras
         mattis consectetur purus sit amet fermentum.</p>

        <h4>Subheading</h4>
        <p>Maecenas sed diam eget risus varius blandit sit amet non magna.</p>
      </div>

      <div class="col-lg-6">
        <h4>Subheading</h4>
          <p>Donec id elit non mi porta gravida at eget metus. Maecenas
          faucibus mollis interdum.</p>

          <h4>Subheading</h4>
          <p>Morbi leo risus, porta ac consectetur ac, vestibulum at eros.
          Cras mattis consectetur purus sit amet fermentum.</p>

        <h4>Subheading</h4>
        <p>Maecenas sed diam eget risus varius blandit sit amet non magna.</p>
      </div>
    </div>

    <footer class="footer">
      <p>&copy; 2016 Company, Inc.</p>
    </footer>
  </div> <!-- /container -->

  <!-- IE10 viewport hack for Surface/desktop Windows 8 bug -->
  <script src="../../assets/js/ie10-viewport-bug-workaround.js"></script>
</body>
```

En toda la aplicación *Sound Challenge* se ha llevado la estructura aplicando las clases a los *div* de *.container- .row- .col-*. Consiguiendo por tanto la adaptación de los elementos a las diferentes pantallas. Por otro lado, también hay que añadir en el *head* del documento HTML un valor de metadata que define que el tamaño de los elementos debe adaptarse a parte de importar los archivos necesarios de Bootstrap para la interpretación de las clases:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

4.6. Control de versiones: Git

Siempre cuando se está desarrollando un software, muchos desarrolladores no expertos se enfrentan a uno de los típicos problemas, guardar las versiones del código evitando que se pueda perder trabajo ya hecho. Muchos caen en realizar muchas copias de seguridad con diferentes nombres teniendo por tanto muchas copias del código siendo esto redundante a la par de peligroso, porque puede ocurrir que se guarde el código con el mismo nombre que en otra versión que se deseaba guardar y se sobrescriba trabajo que no se deseaba perder.

Por otro lado cuando se trabaja en un proyecto con muchos archivos distintos e incluso se trata de un equipo de desarrolladores, se complica aún más la tarea de guardar las copias de seguridad y controlar las versiones de la aplicación. Para ello hoy en día se utiliza tanto de forma personal como profesional una plataforma web que te permite llevar un control de versiones mucho más sencilla, segura y sin tener que ocupar más memoria de la necesaria de forma local, la más famosa es el Git.

Git es un software de control de versiones no lineal, donde se impulsa la ramificación y el trabajo en equipo. Todo software sin importar la complejidad del mismo debería ser controlado por esta aplicación. Además en este proyecto al trabajar con Pycharm se puede configurar el enlace directamente al repositorio personal y por tanto cada cambio que se haga en el código se puede guardar de forma automática desde ahí mismo. Pero también se pueden administrar las versiones mediante comandos tanto en Linux como en Windows.

Para este proyecto se ha utilizado una cuenta en GitLab, que funciona como el GitHub normal pero es privado, está pensado para trabajar de forma privada un grupo de desarrolladores.

Las ventajas que ofrece son las siguientes:

- Acceso remoto al código desde cualquier dispositivo conectado a internet.
- Cada versión que se guarda se compara con la anterior para comprobar qué cambios se han hecho.
- No se sobrescribe ningún archivo, se puede recuperar el proyecto tal como se haya guardado en cada versión o *commit*.
- Se lleva un seguimiento más completo ya que se guarda el día y la hora de la versión guardada además de que se identifica mediante una frase o palabra.
- Muestra gráficas de colaboración entre los desarrolladores que estén modificando un mismo proyecto e identifica a grosso modo el porcentaje de lenguaje de programación que se ha ido utilizando.
- Se pueden tener varias líneas de trabajo en un mismo proyecto para llevar una organización mayor.
- Los archivos locales siempre son la última versión modificada a no ser que se descargue a conciencia una versión en concreto del Git.

La última versión de *Sound Challenge* para la presentación de este trabajo se modificó el 23 de junio de 2017, teniendo en total desde el inicio hasta el final 114 *commits* o versiones guardadas. Por ejemplo, las últimas versiones de la aplicación *Sound Challenge* se muestran a continuación:

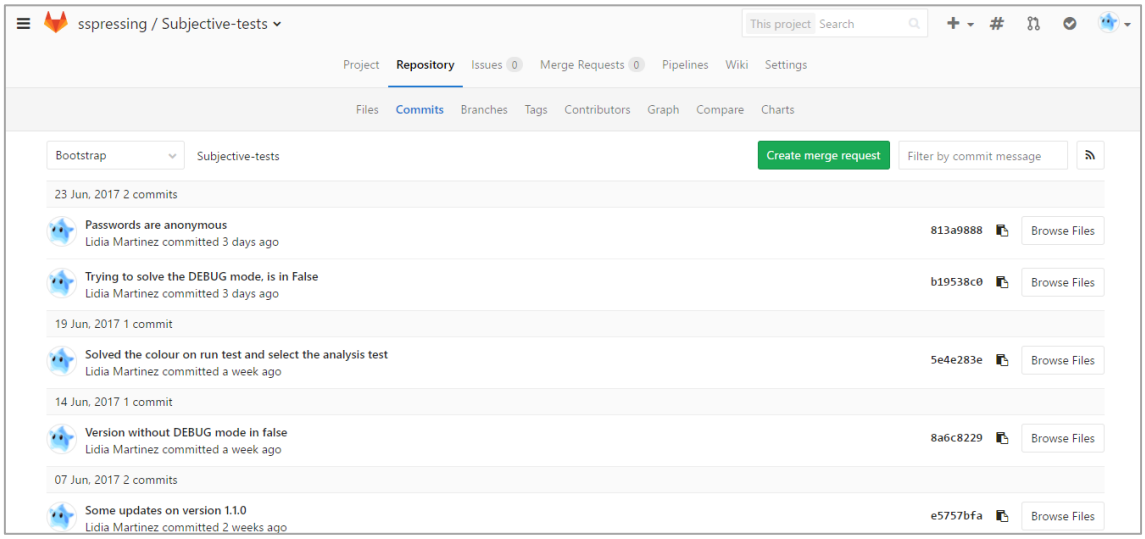


Fig. 12: Últimos commits de Sound Challenge en el GitLab

Gracias a esta herramienta se puede ver el porcentaje de cada lenguaje de programación utilizado como el mostrado en la figura 13. Se puede observar que la mayoría ha sido HTML, pero no es cierto, porque la mayoría ha sido JavaScript, solo que dicho código se ha implementado directamente dentro de los archivos HTML exceptuando algunos archivos externos de JS.

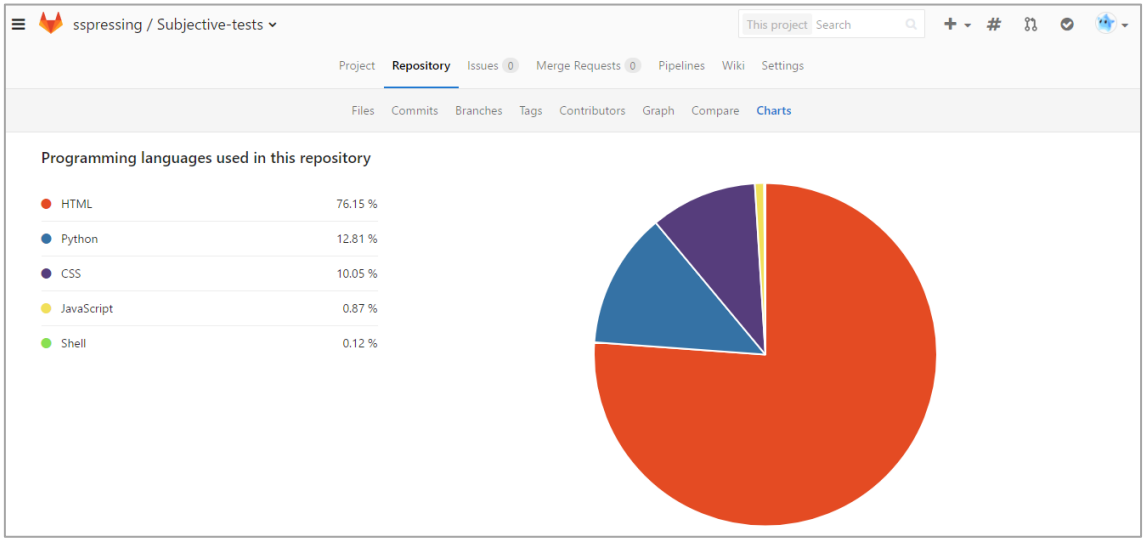


Fig. 13: Porcentaje de lenguajes de programación utilizados en Sound Challenge y mostrados por GitLab

Capítulo 5. Aplicación web: Sound Challenge

Este apartado muestra de forma esquematizada los diferentes macro bloques y sub-bloques que posee la aplicación web Sound Challenge.

Como introducción del capítulo se presenta toda la estructura diferenciando los bloques de la aplicación, relacionando cada parte según pertenezcan a las partes de modelo, vista y controlador (MVC).

Los diferentes sub-apartados están dedicados a explicar cada uno de dichos macro bloques detallando sus funcionalidades.

Termina con los resultados obtenidos de la aplicación de un test de ejemplo realizado por 10 personas.

Capítulo 5. Aplicación web: *Sound Challenge*

El presente capítulo contiene la descripción y especificación del trabajo que se ha realizado en este TFM. El objetivo con este capítulo es presentar la aplicación explicando cada una de las funcionalidades que tiene y cómo están relacionadas siguiendo el proceso de creación de software definido anteriormente. Se muestran los requisitos y especificación definidos, mostrando en el siguiente punto el diseño de la aplicación con todas las funcionalidades en forma de bloques.

5.1. Requisitos y especificaciones de la aplicación

El primer paso en todo desarrollo de software es el análisis de requisitos obteniendo por tanto las especificaciones que guiarán al desarrollador en su trabajo. En *Sound Challenge* como ya se ha comentado, se siguió un ciclo de vida en V donde se fue verificando cada fase; por ello, a medida que se fue desarrollando, se fueron definiendo más especificaciones que necesitaba la aplicación consiguiendo un resultado óptimo para su posterior avance y mantenimiento. A día de hoy la aplicación sigue en desarrollo porque se desean añadir más opciones. Estas opciones están especificadas como líneas futuras.

Los requisitos y especificaciones fueron definidos en colaboración con investigadores del GTAC (Grupo de Tratamiento de Audio y Comunicaciones) de la UPV basándose en un software ya creado en Matlab para realizar las pruebas de escucha. Por tanto las especificaciones para *Sound Challenge* fueron:

I. Requisitos en el inicio de la aplicación:

1. Formulario de introducción de datos del usuario para el análisis del test teniendo información disponible para realizar un filtrado en la evaluación. El registro de usuarios y sus datos se guardará en una tabla de la base de datos.
2. Sistema de bienvenida atractiva que invite a acceder a la aplicación y realizar los tests.
3. Sistema de creación y modificación de pruebas integrado en la aplicación y que sólo puedan acceder a él ciertos usuarios con permisos de administrador.
4. Sistema de análisis de las pruebas integrada en la aplicación web.

II. Creación y modificación de tests:

1. Al añadir audios al diseño, poder seleccionar el trozo de dicho audio para su ejecución.
2. Asegurar que como mínimo hayan dos audios seleccionados y que el número máximo de pares a introducir se rija por esta fórmula: *Nº_audios*(Nº_audios-1)*.
3. Si los audios son de diferente longitud, avisar o no permitir la continuación de la creación del test.
4. Realizar la misma estructura de selección de pares implementada en el software de Matlab: en matriz o en forma consecutiva.
5. Añadir umbral de decisión para excluir jurados automáticamente definiendo valores de probabilidad de repetitividad y consistencia.
6. Añadir una lista con las características a evaluar permitiendo seleccionar algunas por defecto o añadir propias.
7. Corroborar que todos los datos están bien introducidos antes de guardar el diseño con un nombre. El nombre identificará al test de forma única.
8. Cada diseño se guarda en una tabla de la base de datos en forma de texto. El diseño se realizará con objetos.

III. Ejecución de los tests:

1. Sistema central donde se permita acceder a los tests de forma simple.
2. Adquisición de datos de configuración antes de realizar un test. Datos requeridos: tipo de auriculares utilizados y tipo de tarjeta de sonido utilizada.
3. Sistema de entrenamiento del test previo y explicación de las instrucciones de la prueba de escucha para familiarizar al usuario.
4. Ejecución del test simple mostrando las fases en total y siendo guardado el resultado al comprobar que se ha terminado el test. Los pares de sonido deben estar en el mismo orden que en el diseño realizado. Los datos se guardarán en una tabla de la base de datos que está relacionada directamente con los diseños relacionando dichos datos con
5. Dar las gracias al usuario por realizar el test dando en ese momento un *feedback* inmediato.

IV. Análisis de los tests:

1. Implementar los algoritmos de análisis en la misma aplicación.
2. Mostrar con gráficas los datos resultantes.

V. Generales:

1. Realizar un diseño amigable y sencillo de utilizar.
2. La aplicación debe ser ejecutado desde cualquier dispositivo. Realizar un diseño web adaptable.
3. El idioma principal debe ser el inglés.

5.2. Diseño

La siguiente fase del desarrollo de software es la de diseño. En este apartado se va a mostrar dicha fase de la manera más esquemática posible enmarcando y organizando las diferentes partes que tiene la aplicación definida por los requisitos. A continuación se muestra un esquema donde se describe grosso modo las partes fundamentales de la aplicación relacionándolo con el método MVC, comenzando por los modelos creados y siguiendo con la muestra de las vistas explicadas junto a la parte del controlador.

5.2.1. Modelos

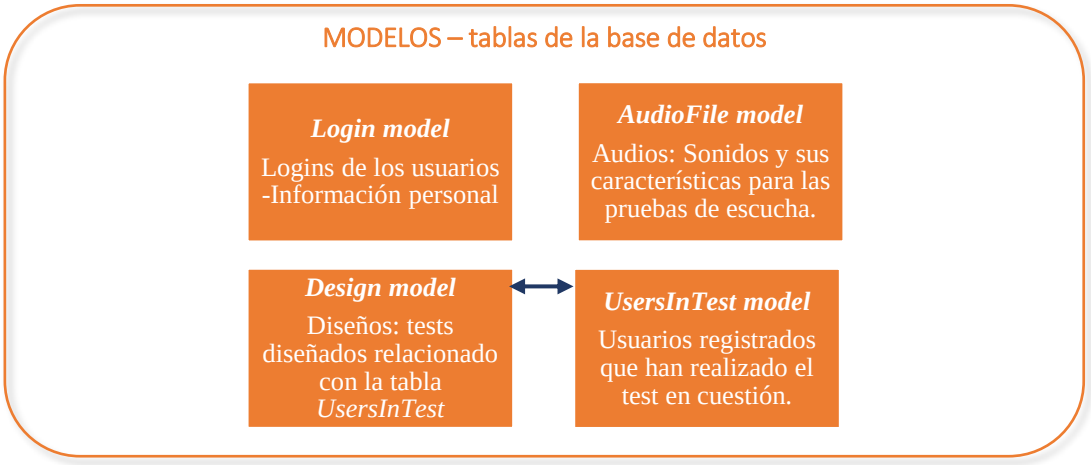


Fig. 14: Modelos de la aplicación Sound Challenge

Como se puede observar en la figura 14, los modelos son cada una de las tablas creadas para administrar los datos de la aplicación. Cada modelo ha sido creado con el propósito de administrar los datos de los usuarios, los audios a utilizar y las pruebas que se van a realizar en la aplicación. Dentro de Django se especifican creando una clase de Python en el archivo *models.py* de la aplicación.

5.2.1.1. Modelo *login* - Registro de datos de usuario

El propósito de este modelo es permitir tener un sistema de registro en la aplicación para acceder a ella con un nombre y contraseña. Por otra parte, en el análisis de datos es importante conocer qué características tienen los usuarios para poder hacer un análisis más concreto de las pruebas. De esta forma los usuarios no tienen que introducir los datos personales cada vez que realizan un test. Los campos que contiene son los siguientes:

Login

- ***Username_email*** (nombre de usuario): Se pide como nombre de usuario den un e-mail para así además de entrar en la aplicación, permitir darle un *feedback* sobre los resultados de los tests en los que ha participado.
- ***Gender*** (género): Se pide que digan si es una mujer o un hombre ya que en los tests de pares a veces resulta revelador las diferencias de opiniones subjetivas que hay entre diferente género.
- ***Age*** (edad): Se guarda la edad del usuario seleccionando un rango ya que igualmente con el género, personas de diferente edad pueden tener opiniones dispares.
- ***Job*** (ocupación): De forma opcional, se guarda datos sobre la ocupación del usuario, de forma que se tiene otro parámetro más para filtrar resultados.
- ***Country*** (país): También de forma opcional, los usuarios pueden introducir el país de procedencia y de esta manera se podría conocer desde cuantos países se han registrado en la aplicación.
- Campo no visible ***Admin***: Es un valor booleano de *true* o *false*. Indica si el usuario tiene permisos de administrador o no y solo se puede modificar dentro de la adminstración de Django (figura 17).

Fig. 15: Campos del modelo de registro de usuarios de Sound Challenge

Cuando se registra un nuevo usuario, en la versión actual de la aplicación, se guardan los datos tanto en ese modelo como en al administrador de usuarios interno de Django para realizar la autentificación de estos de forma anónima y segura. La contraseña del usuario solo se guarda mediante encriptación en dicho registro de usuarios de Django, para asegurar de esta manera que no se viola ningún derecho de privacidad del usuario. El modelo implementado en *models.py* y su administración en la página de Django de administrador son de la siguiente manera.

```
class Login(models.Model):

    username_email = models.EmailField(max_length=200)
    gender = models.CharField(max_length=100)
    age = models.CharField(max_length=100)
    job = models.CharField(max_length=100, blank='true')
    country = models.CharField(max_length=100, blank='true')
    admin = models.CharField(max_length=10)

    def __str__(self):
        # para representar la variable de nombre de usuario
        # en formato string por la linea de comandos
        return self.username_email
```

Fig. 16: Clase del modelo Login implementado en Django

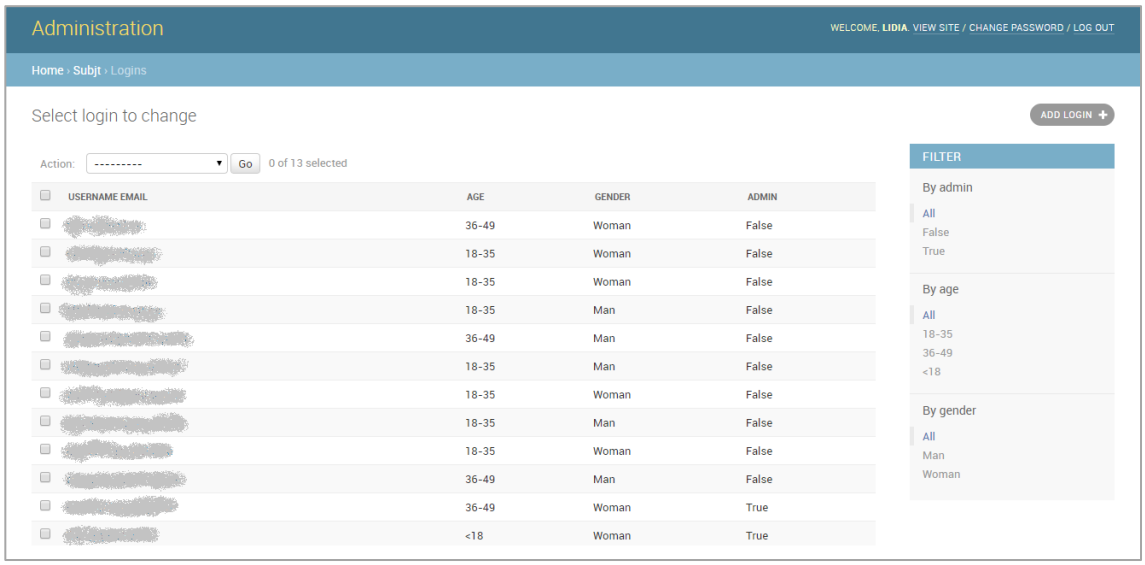


Fig. 17: Gestión de la tabla de registros en la administración de Django (usuarios registrados censurados)

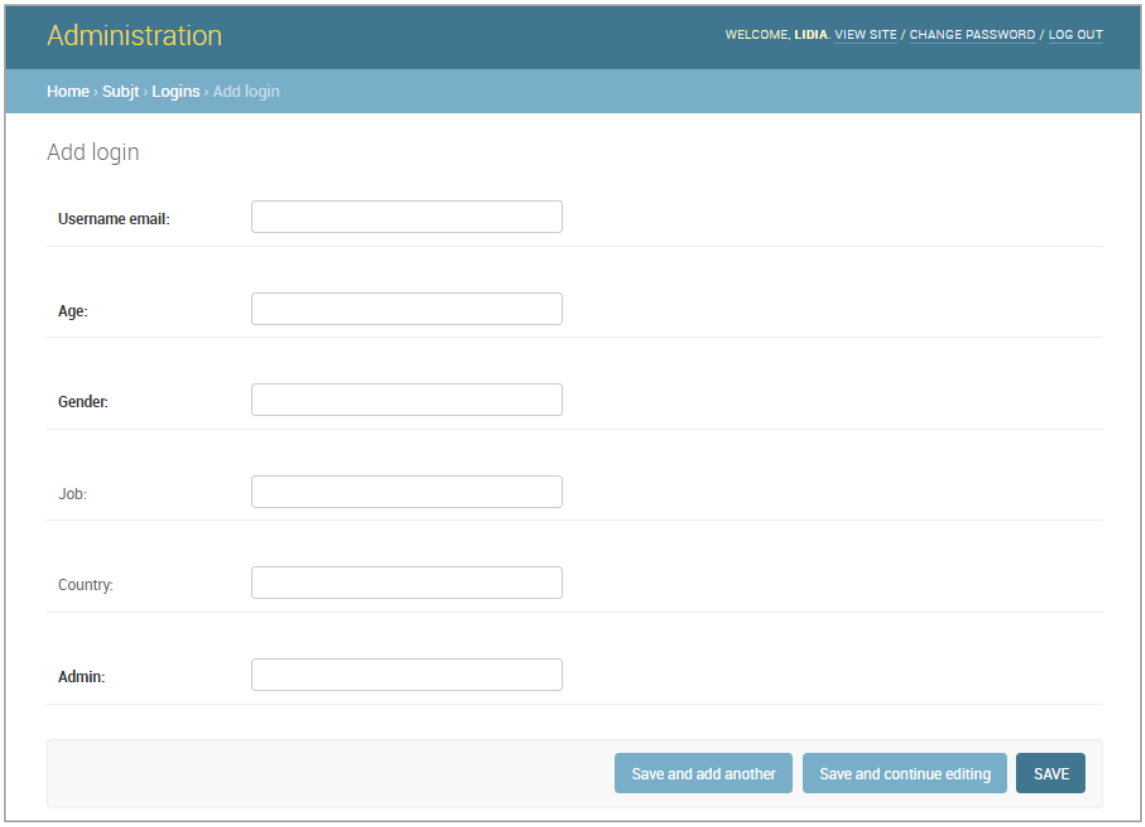


Fig. 18: Añadir un nuevo usuario en el modelo Login de Sound Challenge

5.2.1.2. Modelo *AudioFiles* – Almacenamiento de audios

Esta tabla o modelo está implementada con la finalidad de servir audios a la aplicación para diseñar los tests y ejecutarlos. Es importante no eliminar los audios sin conocer si se han seleccionado para algún diseño de prueba. Los audios se guardan dentro del proyecto de Django en una carpeta estática, cada vez que se carga un audio se sube al servidor. Los campos que tiene esta tabla son los siguientes:

AudioFiles

- ***File_name*** (nombre del archivo de audio): A cada audio se le identifica por el nombre que se le introduce en este campo. Es el que aparecerá en la parte de creación de pruebas.
- ***Audio_type*** (tipo de audio): Campo de texto para realizar un filtrado más concreto. El/la diseñador/a introducirá mediante una palabra o una frase qué tipo de audio es (ej.: música, ruido, etc.)
- ***Processing_type*** (tipo de procesado): Campo de texto para organizar los audios según su tipo de procesado. También lo deberá introducir el/la diseñador/a a mano.
- ***File_code*** (código de archivo): Campo de texto para tener una identificación más corta que con el nombre. Normalmente el código que se aplica son siglas relevantes al nombre.
- ***File*** (archivo): Es un campo tipo *MediaFile* donde se sube el archivo de audio en concreto. Los archivos deben ser de extensión *.mp3* porque dicho formato es compatible con todos los navegadores de hoy en día.

Fig. 19: Campos del modelo de almacenamiento de audios en Sound Challenge

A continuación se muestra el modelo implementado en una clase en *models.py* de Django donde se especifican dos funciones para hacer posible la eliminación de los archivos de audio por completo, que se eliminen tanto de la tabla como del directorio estático.

```
class AudioFile(models.Model):

    file_name = models.CharField(max_length=100)
    audio_type = models.CharField(max_length=100)
    processing_type = models.CharField(max_length=100)
    file_code = models.CharField(max_length=100)
    file = models.FileField()

    def __str__(self):
        return self.file_name


# These two auto-delete files from filesystem when they are unneeded:
@receiver(models.signals.post_delete, sender=AudioFile)
def auto_delete_file_on_delete(sender, instance, **kwargs):
    """Deletes file from filesystem
    when corresponding `MediaFile` object is deleted.
    """
    if instance.file:
        if os.path.isfile(instance.file.path):
            os.remove(instance.file.path)
```

Fig. 20: Código de pyhton del modelo AudioFiles de Sound Challenge (1a parte)

```python
@receiver(models.signals.pre_save, sender=AudioFile)

def auto_delete_file_on_change(sender, instance, **kwargs):
    """Deletes file from filesystem
    when corresponding `MediaFile` object is changed.
    """
    if not instance.pk:
        return False

    try:
        old_file = AudioFile.objects.get(pk=instance.pk).file
    except AudioFile.DoesNotExist:
        return False

    new_file = instance.file
    if not old_file == new_file:
        if os.path.isfile(old_file.path):
            os.remove(old_file.path)
```

Fig. 21: Código de Python del modelo AudioFiles de Sound Challenge (2a parte)

Dentro de la administración interna de Django el modelo se gestiona mostrando la apariencia de las figura 22 y 23. Desde ahí se pueden añadir, editar y modificar.

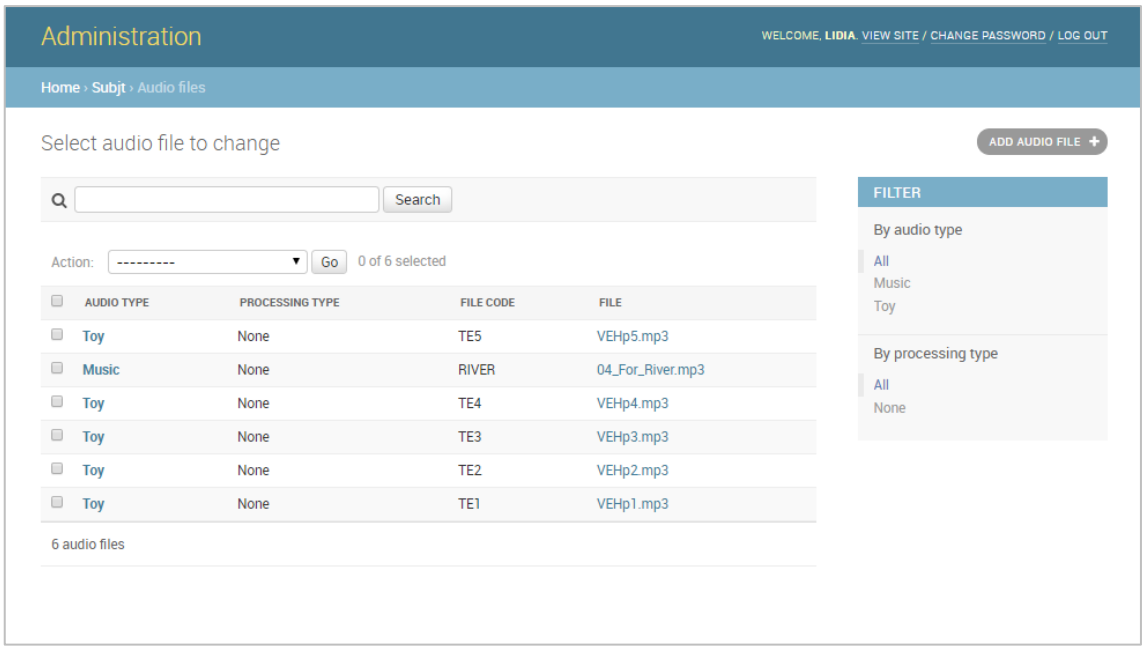


Fig. 22: Modelo de archivos de audio de Sound Challenge en la administración de Django

En la figura 23 se muestra el medio por el cual habría que añadir los audios a la base de datos. Como ya se ha comentado en la especificación del campo del archivo, se debe utilizar audios *.mp3* por cuestiones de compatibilidad entre los navegadores, pero resulta que este contenedor se genera mediante compresión con pérdidas, por ello también en la figura 23 se pide que la tasa de compresión aplicada sea como mínimo de 320 kbps ya que a partir de dicha tasa la diferencia entre un archivo *.wav* al *.mp3* no son relevantes para el oído humano.

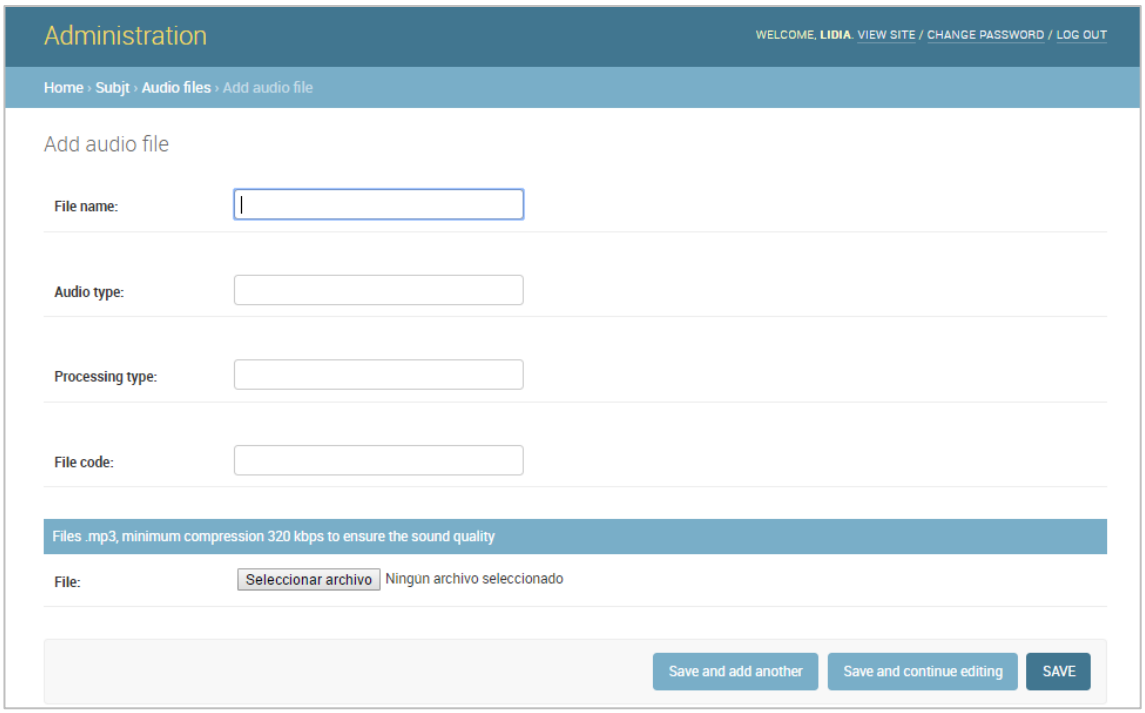


Fig. 23: Añadir un nuevo archivo de audio al modelo de Sound Challenge

5.2.1.3. Modelo *Design* – Almacenamiento de las pruebas diseñadas

El modelo de diseños está implementado con el objetivo de almacenar las pruebas diseñadas enlazando los usuarios que hayan hecho cada prueba. Está pensado para que se pueda identificar a cada prueba de forma unívoca con su nombre. La configuración de los pares, los audios y sus características se almacenan en forma de texto habiéndolo creando previamente mediando objetos con JavaScript en la parte del cliente, se muestran más adelante. Los campos que tiene son los siguientes:

Design

- ***name*** (nombre del archivo del diseño o prueba): Nombre identificador del diseño que aparecerá en la lista de pruebas de escucha disponible.
- ***list_of_sounds*** (lista de audios): Lista de audios provenientes del modelo de *AudioFiles* configurados para el test en cuestión. Las características de duración y su nombre se almacenan en forma de texto siendo previamente ordenado como un objeto.
- ***sound_data*** (datos sobre el diseño): Texto que define el número de comparaciones, la configuración de repetitividad y consistencia, y la combinación de los sonidos en pares en el órden diseñado. Toda esta configuración previamente se genera creando un objeto.
- ***par_data*** (lista de parámetros subjetivos a evaluar): Vector guardado en forma de texto sobre los parámetros a evaluar en la prueba de escucha.
- ***eiditing*** (campo de verfificación): Campo extra creado para definir si un test no realizado por ningún usuario está modificándose o no. Si se está editando no aparece en la lista de pruebas de escucha disponible.

Fig. 24: Campos del modelo de diseños de Sound Challenge

El código en Python que se encarga de crear el modelo de diseños mediante una clase, es el siguiente:

```
class Design(models.Model):

    name = models.CharField(max_length=200)
    list_of_sounds = models.CharField(max_length=9000000,default='')
    sound_data = models.CharField(max_length=9000000)
    par_data = models.CharField(max_length=9000000)
    editing = models.CharField(max_length=100,default='NO')

    def __str__(self):
        return self.name
```

Fig. 25: Código de python del modelo de diseños de Sound Challenge

En la administración de Django solamente es útil manejar esta tabla para consultar qué datos tiene y eliminar algún diseño si se desea. La lista de diseños es la figura 26 y la información de uno de los diseños ya creados y testeados es la figura 27.

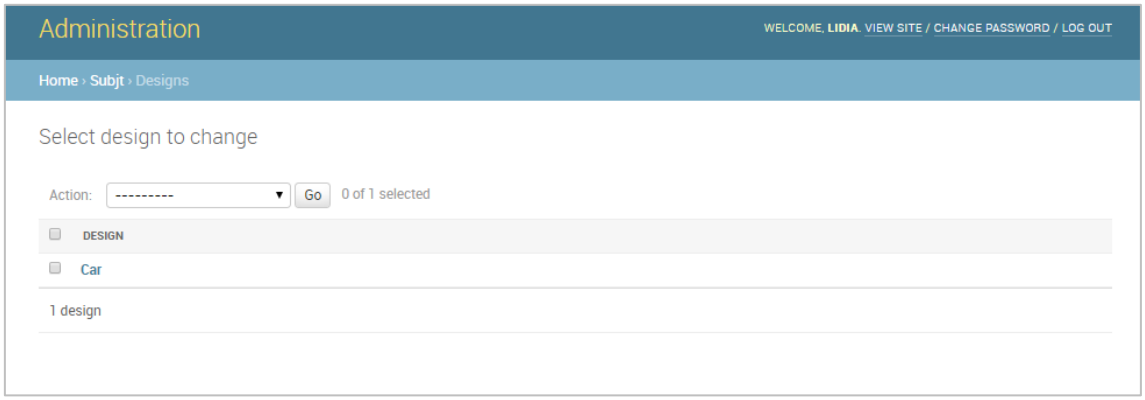


Fig. 26: Lista de diseños de Sound Challenge en el administrador de Django

De momento solo hay un test en la base de datos, es el test de prueba teniendo varios usuarios que lo han hecho. En la figura 27 se puede observar cómo están tratados los datos de configuración, es todo texto con la edición en administración desactivada. Cabe destacar que este diseño tiene el estado de no edición, esto es que, aparecerá en la lista de pruebas disponibles.

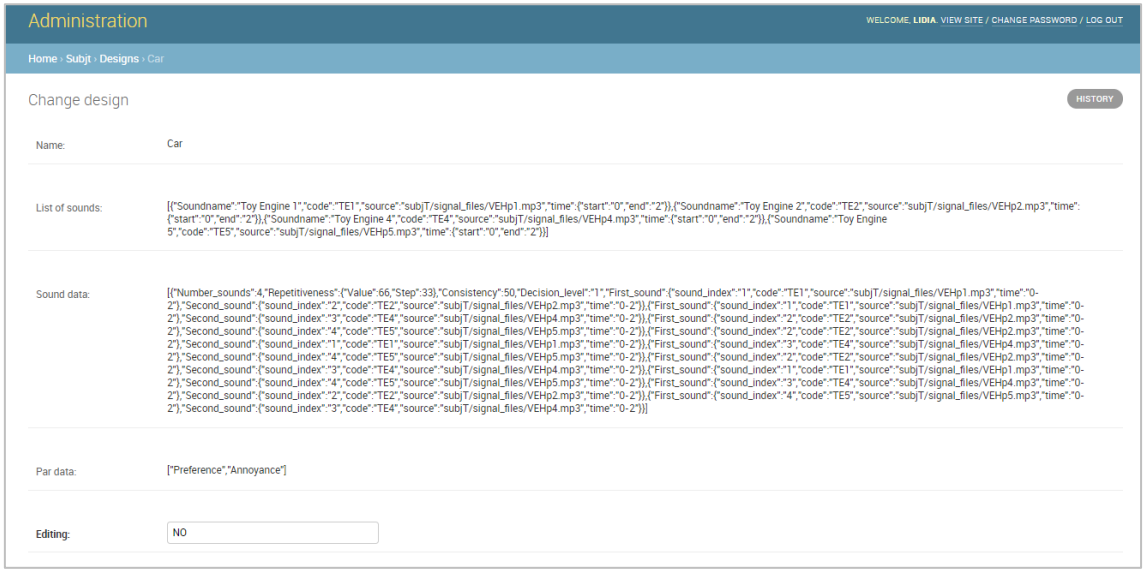


Fig. 27: Campos del diseño de prueba 'Car' dentro de la administración de Django

5.2.1.4. Modelo *UsersInTest* – Tabla de usuarios que han hecho las pruebas

Finalmente, el último modelo implementado es el que registra qué usuarios han realizado el test guardando por tanto sus respuestas y la información relevante a su configuración de elementos a la hora de realizarlo. Esta tabla relaciona con la de diseños mediante el número *pk* que tiene cada uno de los tests. En la administración de Django se visualizan los usuarios dentro de cada prueba como una segunda tabla. Primero indiquemos qué campos posee:

UsersInTest

- ***test*** (enlace al la prueba en cuestión): Elemento denominado *foreignKey* para enlazar el usuario al test.
- ***email*** (nombre de usuario del juez): Nombre de usuario para registrar las respuestas identificándolo univócamente.
- ***data*** (datos sobre el usuario en el test): Datos del usuario sobre la configuración de los auriculares y tarjeta de audio, además de las respuestas elegidas junto a información sobre el día, la hora y el lugar que se hizo el test. También se adquieren datos del navegador desde donde se ha realizado. Todo ello en formato texo previamente creado mediante un objeto.

Fig. 28: Campos del modelo UsersInTest de Sound Challenge

Dentro del administrador de Django se puede decidir eliminar un usuario que por ejemplo no es relevante para el estudio. La figura 29 es la continuación del diseño 'Car' mostrado en la figura 27, ya que ahí es donde se visualizan los usuarios. Los datos personales de usuario como el e-mail y su dirección IP se han censurado.

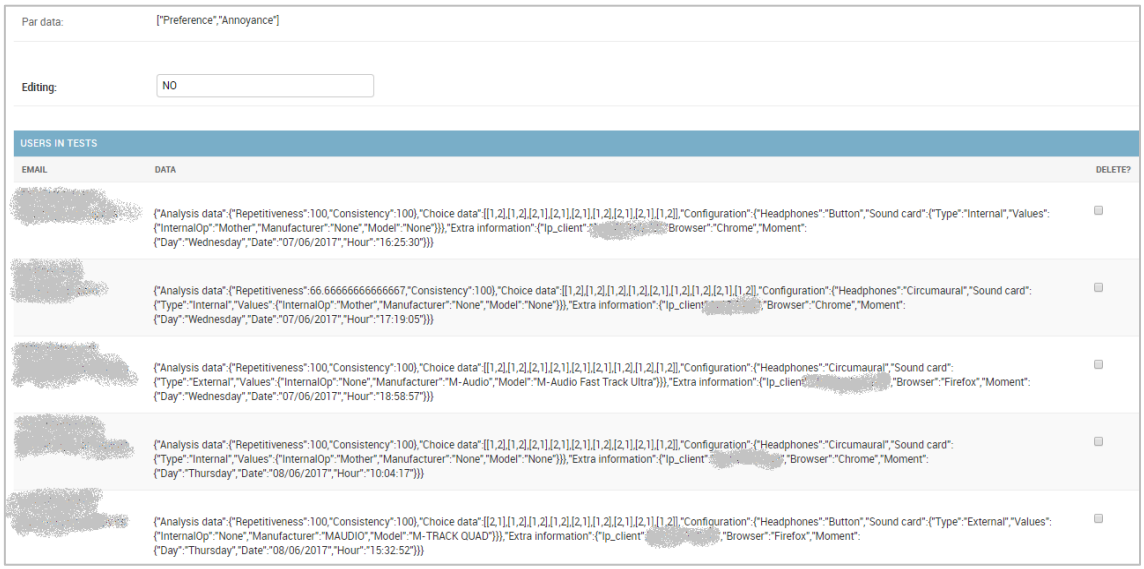


Fig. 29: Usuarios en el test 'Car' de ejemplo de Sound Challenge dentro del administrador de Django

5.2.2. Vistas y controlador

En el presente apartado se van a mostrar y explicar las partes más relevantes de la aplicación mediante las vistas, las cuales son a las que accede y ve el usuario final. El método de explicación va a ser analizar cada uno de los archivos *.html* (vistas) destacando las partes del código que realiza la comunicación con los modelos, es decir, mostrando el controlador en cada caso. El orden de las vistas a mostrar va a ser como el flujo de trabajo mostrado en la figura 30, siguiendo el orden impuesto por los números y siendo un usuario administrador (modificando el permiso desde Django anteriormente mostrado).

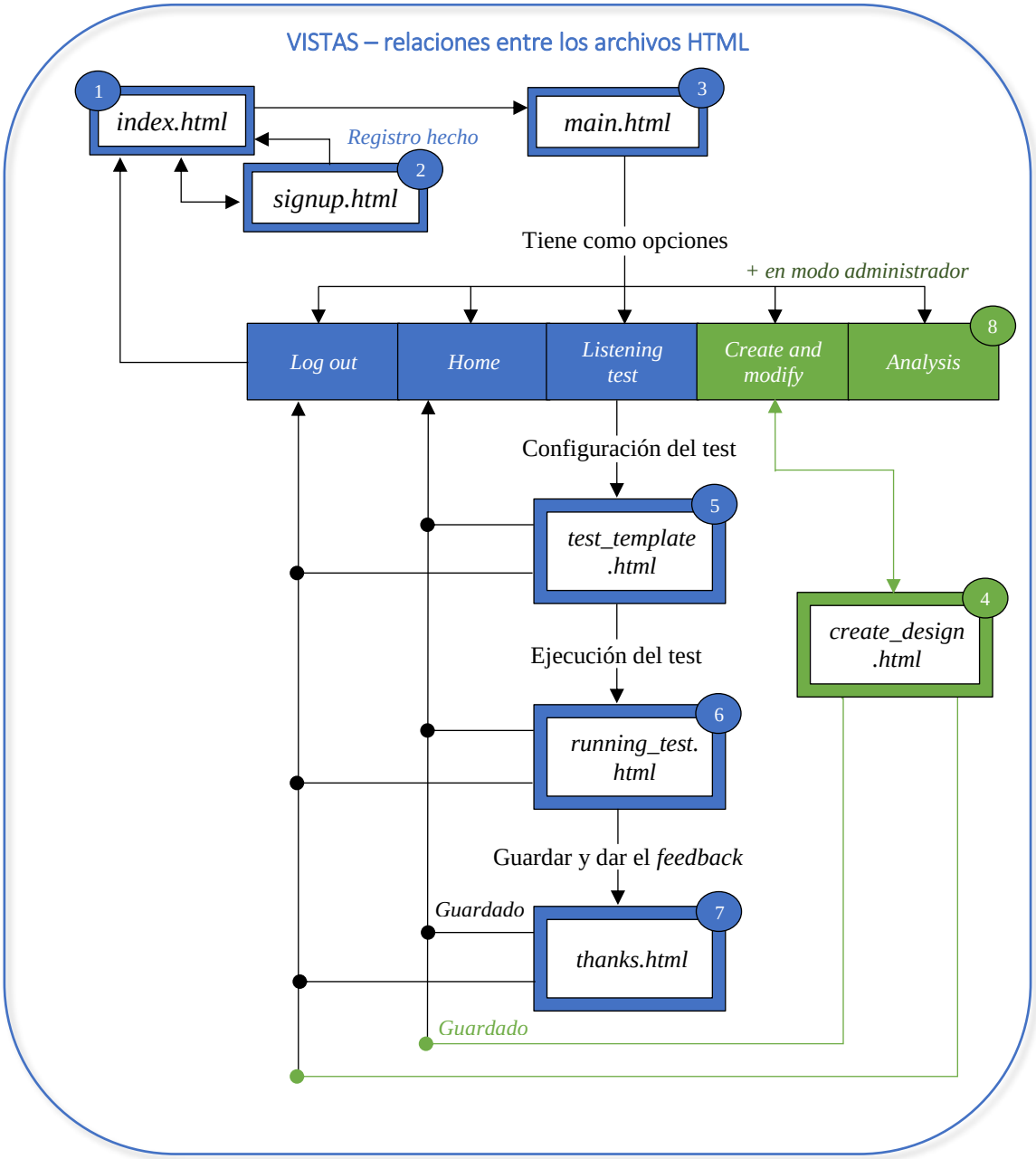


Fig. 30: Esquema de las vistas de la aplicación web Sound Challenge

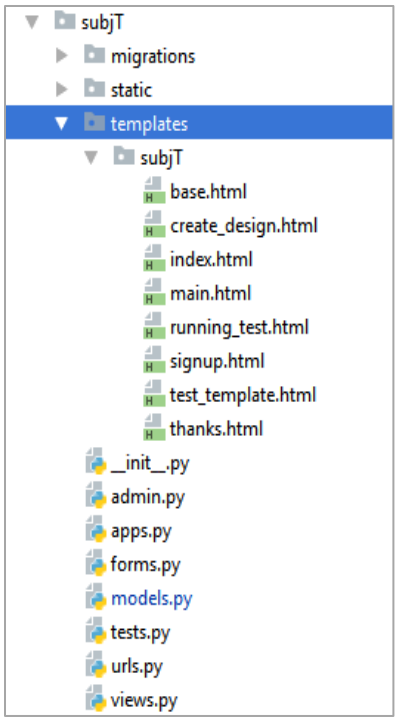


Fig. 31: Carpeta de la aplicación de Sound Challenge dentro de Django

Los elementos *.html* mostrados en la figura 30 dentro del proyecto Django se almacenan en una carpeta denominada *templates* dentro de la carpeta de la aplicación de test subjetivos (figura 31).

En ella aparte de aparecer todos los archivos HTML antes nombrados en la figura 30, aparece un archivo denominado *base.html*. En Django se puede extender el contenido y estilo de un código HTML a otro para evitar estar repitiendo código.

En concreto *base* se ha utilizado con la finalidad de introducir los elementos comunes de las vistas dentro de la aplicación. Toda la programación HTML5, CSS3 y JavaScript simplemente se va a mostrar mediante el resultado final enseñando las capturas de las vistas ya que si no la presente memoria tendría una longitud mucho mayor e innecesaria.

Por tanto siguiendo el orden marcado en la figura 30, a continuación se va a explicar cada vista y su controlador. Las imágenes de la aplicación mostradas son tanto en formato de PC mediano como en formato móvil para ver las diferencias de estilo ejecutándolo con el navegador Google Chrome.

5.2.2.1. Vista de la página de entrada – *index.html*

Sound Challenge a día de hoy sigue en proceso de desarrollo por ello para hacer las pruebas se ejecuta en dominio local haciendo como servidor el mismo PC con el que se crea la aplicación. Por ello al acceder a la página en las capturas de a continuación aparece como URL:

localhost:8000/subjT

Se accede de forma local mediante el puerto 8000 a la aplicación denominada *subjT*. La primera vista que se encuentra el usuario por tanto es la de acceder a la aplicación con un nombre de usuario ya creado o la de registrarse.

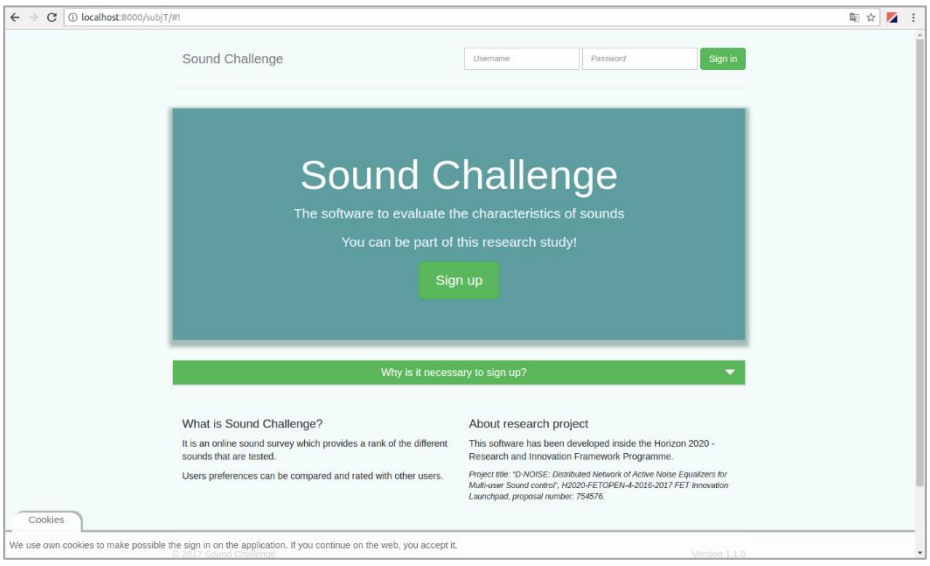


Fig. 32: Vista inicial de Sound Challenge

Para acceder a la vista con la introducción de la anterior URL, Django consulta la lista de URL y lo redirige al controlador que carga el archivo HTML *index.html*. Estas instrucciones se encuentran en el archivo *views.py*, en concreto para esta vista el controlador es el siguiente:

```
def Index(request):

    return render(request, 'subjT/index.html')
```

Fig. 33: Código en python del controlador de index.html

La vista en formato móvil tiene este aspecto:

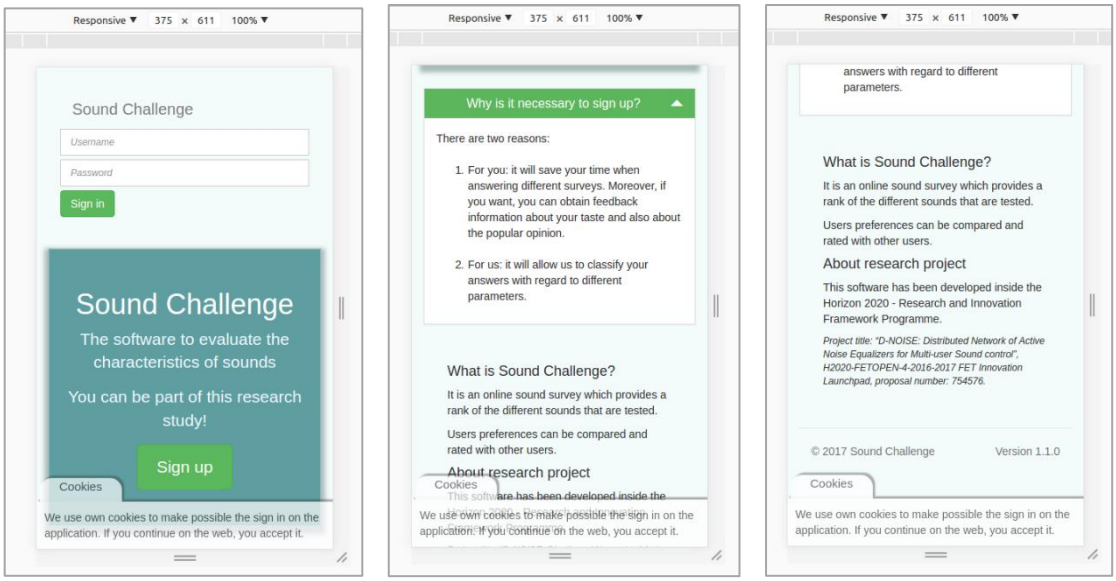


Fig. 34: Inicio de Sound Challenge en formato móvil

Las únicas acciones en esta vista son las de registrarse (*sign up*) y de entrar en la aplicación (*sign in*) donde ambas son formularios para acceder a cada uno de los controladores correspondientes que acceden a las vistas adecuadas.

5.2.2.2. Vista del formulario de registro de nuevos usuarios – *signup.html*

Si se sigue el recorrido que haría un nuevo usuario en la aplicación, el siguiente paso sería registrarse pulsando en el botón de *Sign up* mostrado en las figuras 33 y 34. Al pulsar en él, el formulario del botón envía una petición a la URL: *localhost:8000/subjT/login*, definido con el siguiente código de Python:

```
def GetLogin(request):

    return render(request, 'subjT/signup.html',{
        'message':'',
    })
```

Fig. 35: Código en python del controlador para acceder al registro

La siguiente vista que aparece es el formulario para introducir los datos que se guardarán en el modelo de *login*. Como se puede ver en las figuras 36 y 37, solo es requerido en email como nombre de usuario, el género y la edad, los otros dos campos son opcionales. Para cumplir la ley, a la derecha está disponible la explicación de que los datos serán guardados confidencialmente y que se administrarán solo para el propósito de realizar análisis subjetivo, además de indicar en la vista inicial (figuras 33 y 34) que las cookies que se utilizan son con el propósito de iniciar la sesión del usuario en la aplicación.

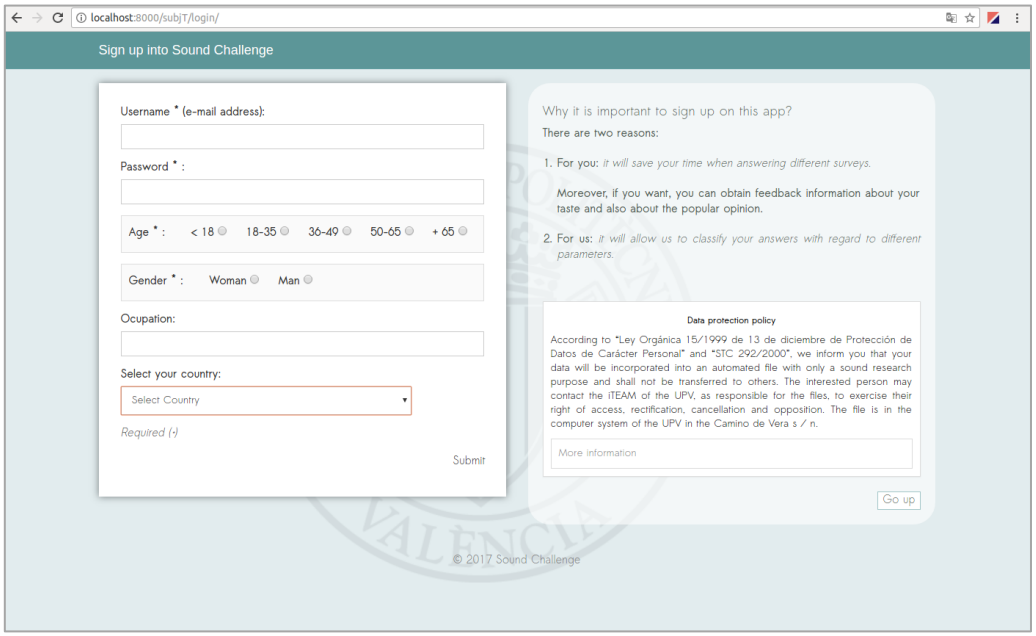


Fig. 37: Vista del registro de un nuevo usuario de Sound Challenge

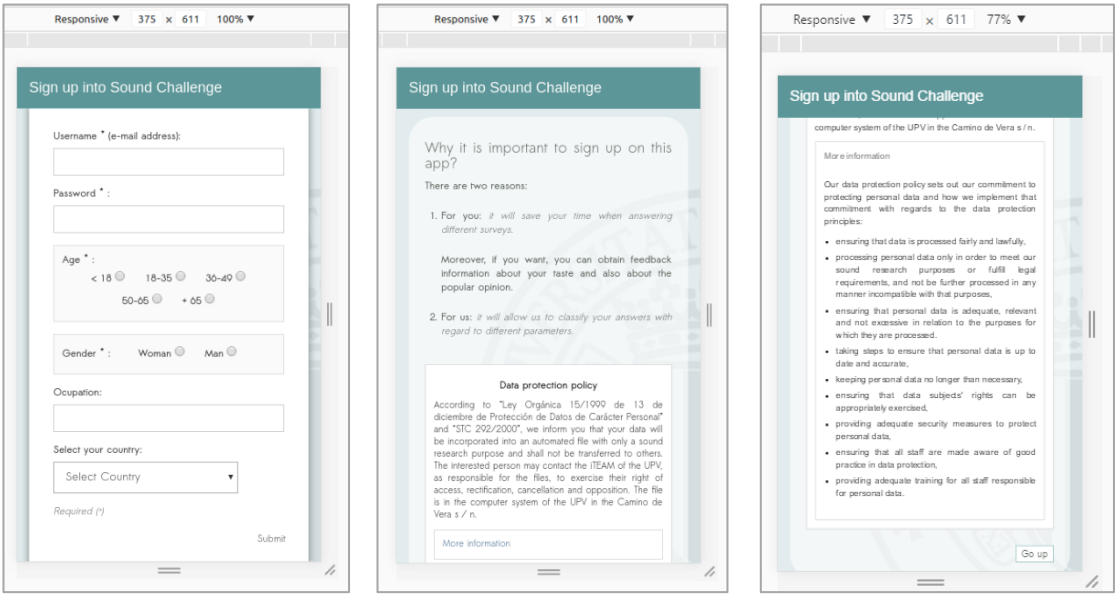


Fig. 36: Vista del registro de usuarios en formato móvil

Una vez el futuro usuario introduce los datos del registro, se envía el formulario haciendo la acción de POST llegando otra vez al controlador en *views.py* accediendo esta vez a la siguiente función de la figura 38. De esta forma se enlaza vista con modelo, modificando el modelo *login*.

```
def login_main(request):
    try:
        if request.method == 'POST':
            form = LoginForm(request.POST)
            if form.is_valid(): # Se valida el formulario
                cd = form.cleaned_data

                f = request.POST
                same_login =Login.objects.filter(username_email=f['username_email'])
                same_login_user = User.objects.filter(username=f['username_email'])
                # De momento si existe un usuario con el mismo nombre de usuario se
                   sobreescribe
                if same_login.exists():
                    same_login.delete()

                if same_login_user.exists():
                    same_login_user.delete()

                # Se crea un nuevo usuario dentro de la clase tipo USER de Django
                user = user.objects.create_user(username=f['username_email'],
                                                email=f['username_email'])
                user.set_password(f['password'])
                user.save()

                # Registrar los valores del formulario en cada campo del modelo
                   login

                obj = Login()# Se carga el modelo de Login definiendo así un objeto

                obj.username_email = f['username_email']
                obj.age = f['age']
                obj.gender = f['gender']
                obj.job = f['job']
                obj.country = f['country']
                obj.admin = f['admin']
                # for key, value in cd.iteritems():
                #     setattr(obj, key, value)
                #     request.session[key] = cd[key]
                obj.save()

            return render(request, 'subjT/index.html', {
                'if_admin': 'No',
            }, messages.warning(request, 'Your signed up successfully!'))

        else:# Si no es un POST se vuelve al índice
            return HttpResponseRedirect('/subjT')
    except: # Si existe un error se salta al índice también
        return HttpResponseRedirect('/subjT')
```

Fig. 38: Código en python del controlador de la vista de registro de usuarios

Cuando se guarda correctamente el nuevo usuario, se vuelve al índice mostrando un mensaje de que se ha registrado con éxito. Si no se realiza un nuevo registro, se puede volver a la página del índice yendo para atrás en el navegador, en esa acción no actuaría ningún controlador.

5.2.2.3. Vista de la pantalla principal de la aplicación – *main.html*

Una vez el usuario está registrado en la aplicación el siguiente paso es acceder a la misma introduciendo el nombre y contraseña. Si no existe el usuario o no es válido, aparece un mensaje indicándolo y no permite entrar en la aplicación.

Cuando se envía el formulario presionando el botón de *sign in* mostrado en las figuras 32 y 34, se envía el formulario al controlador comprobando que se pueden guardar cookies, ya que se utilizan para definir la sesión. Estas cookies se borran al cerrar la aplicación. Tiene el siguiente código para mostrar o no la vista principal de *Sound Challenge*.

```
# Autentificación del usuario con el método de python
user = auth.authenticate(username=form['username_email'],
password=form['password'])

if user is not None and user.is_active:

    obj2 = Login.objects.filter(username_email=form['username_email'],admin='True')
    if obj2.exists():
        adm = 'Yes'
    else:
        adm = 'No'

    # auth.login(request, user)

    return render(request, 'subjT/main.html',{
        'if_admin': adm, 'username': 'User: ' + form['username_email'] ,
        'email': form['username_email'],

    })

else: # Si no concuerda, se devuelve a la URL del índice con un mensaje
    messages.warning(request, 'Insert a correct email and password')
    return HttpResponseRedirect('/subjT')
```

Fig. 39: Código de python del controlador para acceder a la vista principal de Sound Challenge

Se supondrá que el usuario iniciado en la aplicación tiene permisos de administrador, esto es que podrá acceder a las opciones de creación, modificación y análisis de las pruebas. El permiso de administrador está reservado para los investigadores que vayan a diseñar las pruebas y los que están interesados en conseguir datos de los audios. Por tanto, la página principal aparece con la opción del *Home* seleccionada.

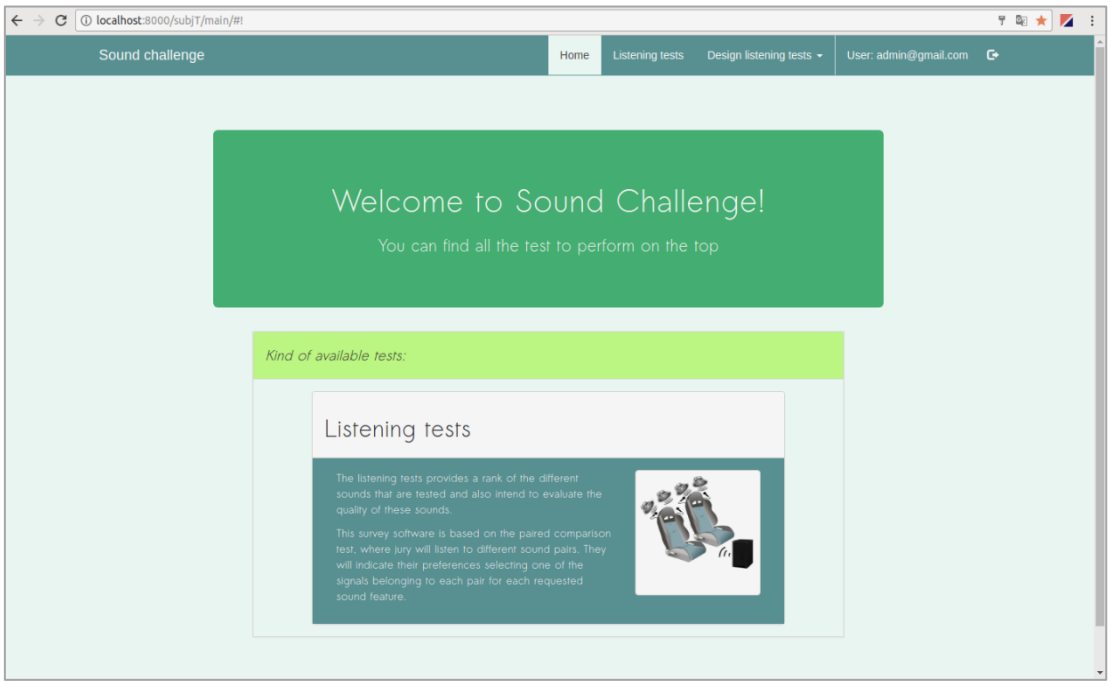


Fig. 40: Vista principal en formato de PC - Home

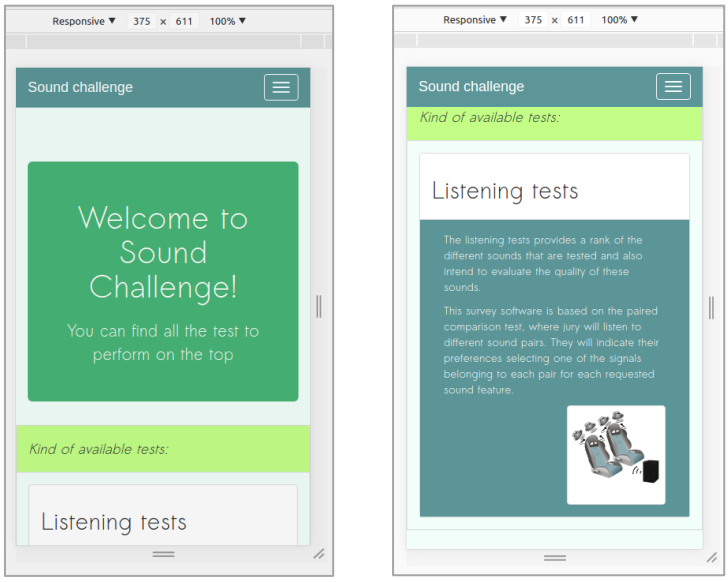


Fig. 41: Vista principal en formato móvil - Home

El usuario de *admin@gmail.com* no es un correo real, en un futuro se implementará la verificación del correo electrónico para realizar un *feedback* adecuado, pero por ahora no está realizado.

Dependiendo de las opciones disponibles en el navegador superior, se accede a uno u otro hilo del programa. Los siguientes puntos son la descripción de dichas opciones definiendo el código del controlador que corresponda para cada caso. Destacar que donde aparece el nombre y el icono a la derecha en el navegador, es para salir de la aplicación (*log out*).

5.2.2.4. Vista de creación de diseños – *create_design.html*

Suponiendo que no existiera ningún test de escucha (*listening test*), el primer paso ya que el usuario es administrador, es crear un diseño desde cero. Para ello se pulsa en la opción del navegador denominada *Design listening test* y eligiendo dentro de esta, la opción de crear y modificar como se muestra en la figura 43.

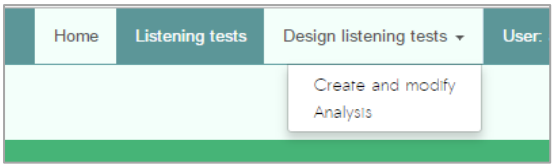


Fig. 42: Selección de la opción de crear y modificar tests

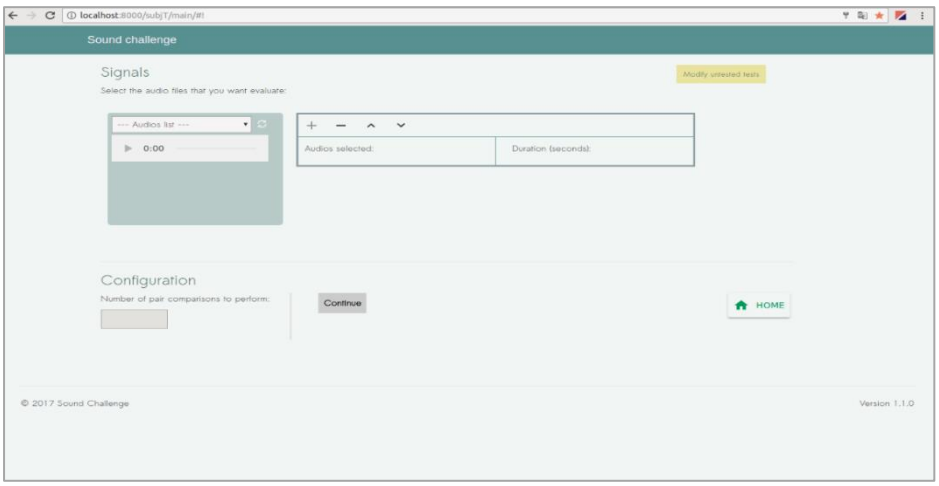


Fig. 43: Vista principal - Creación y modificación de pruebas

Como se puede observar en la figura 42, en la misma vista principal, la parte central se cambia para dar paso a la selección de audios para el nuevo diseño con la condición de que si se vuelve al *home*, todos los avances se perderán.

La lista de audios mostrada en la parte izquierda, se carga de forma asíncrona accediendo a la tabla de audios de la base de datos mediante un método AJAX. Se realiza una petición GET para cargar todos los audios por si alguno se ha modificado, también el botón de al lado de la lista realiza la misma petición (figura 43).

```
// Function to get all the signals on the BD:
function get_signals(callback){
    //Consulta con AJAX
    var signals ='';
    $.ajax({
        url: "{% url 'subjT:get_data_signals' %}",
        type:'get',
        success: function (response) {
            //lo que haces si es exitoso
            var aux = response['ser_signals'];
            signals = JSON.parse(aux.replace(/&quot;/g, '"'));

            callback(signals)
        }
    });

    return signals;
}
```

Fig. 44: Parte del controlador de la vista principal - petición audios AJAX

Suponiendo que se seleccionan dos audios con la misma duración se tendría como máximo dos posibles parejas:

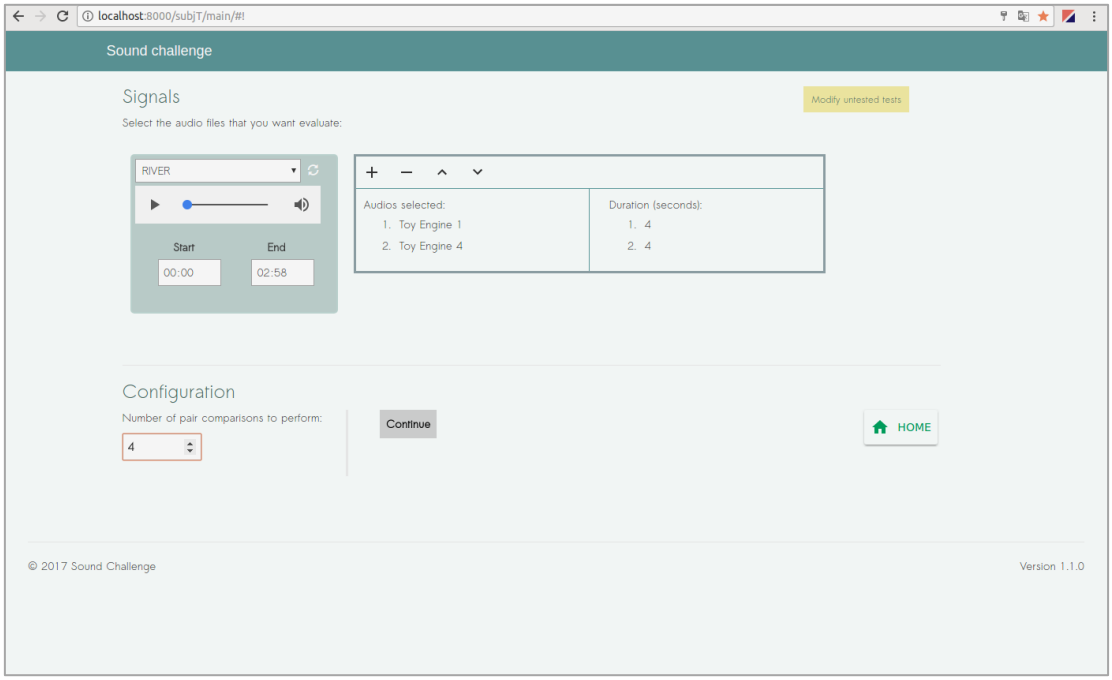


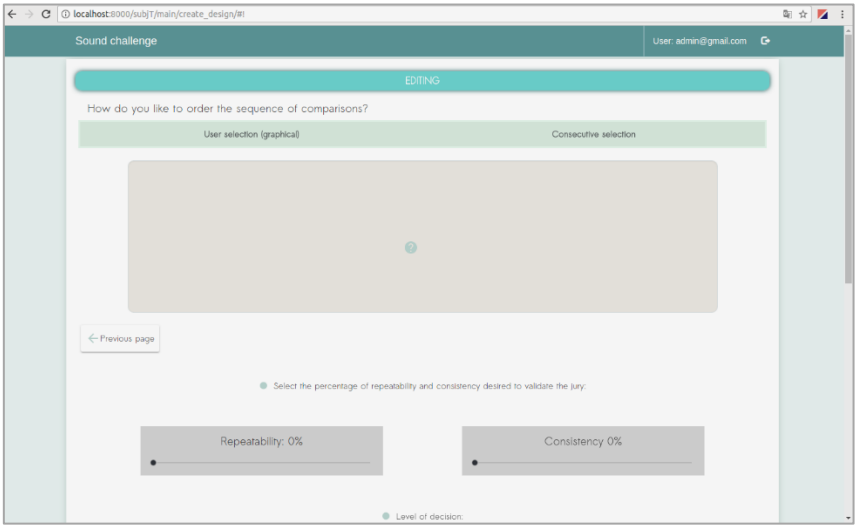
Fig. 45: Selección de audios de ejemplo - Vista principal

En el ejemplo mostrado en la figura 45 se ha escrito 4 comparaciones (no deja continuar porque el máximo son 2). Por tanto si se especifican 2 y se da a continuar, la petición es un formulario que se envía al correspondiente controlador que redirige la petición a la vista de creación. Hay dos situaciones: que se vaya a crear de nuevo el test o que se edite uno ya existente, todo ello se especifica con instrucciones *if…else* dentro del controlador:

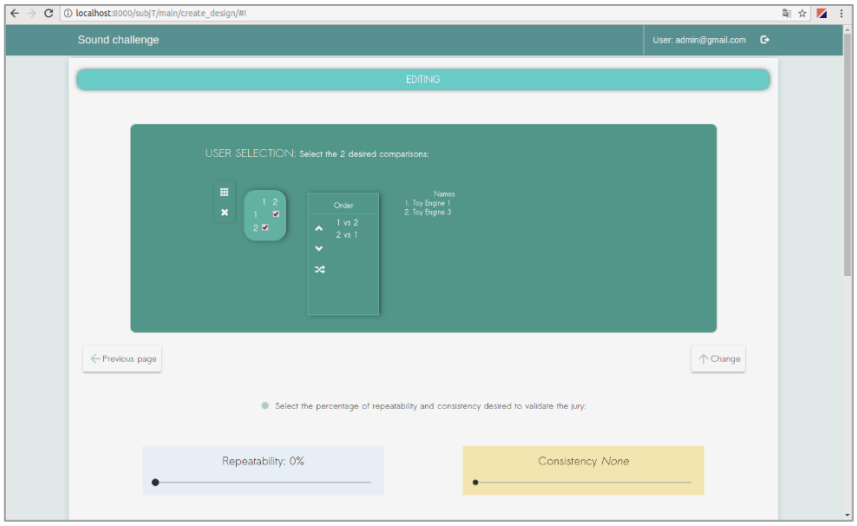
```
def step2_create(request):

    designs = Design.objects.all()
    if request.method == 'POST':

        form_create = request.POST
        audio_selected = form_create['value_list'].encode('utf8')
        num_comp = form_create['num_comp']
        username = form_create['username']
        email = form_create['email']
        editing = form_create['editing']
        if editing != 'NO':

            if Design.objects.filter(name=editing).exists():
                obj_editing = Design.objects.get(name=editing)
                obj_editing.editing = 'YES'
                obj_editing.save()

                return render(request, 'subjT/create_design.html', {
                    'audio_selected': audio_selected,
                    'num_comp': num_comp,
                    # 'sec_longer': sec_longer,
                    # 'signal_list': signals,
                    'username': username,
                    'isediting': editing,
                    'list_designs': designs,
                    'email': email,
                })
            else:
                # If the design doesn't exists
                return render(request, 'subjT/main.html', {
                    'if_admin': form_create['adm'], 'username':
                         form_create['username'],
                    'test_list': designs, 'email': form_create['email'],
                })
        else:

            return render(request, 'subjT/create_design.html', {
                'audio_selected': audio_selected,
                'num_comp': num_comp,
                'username': username,
                'isediting': editing,
                'list_designs': designs,
                'email': email,
            })
    else:
        return HttpResponseRedirect('/subjT')
```

Fig. 46: Controlador para acceder a la vista de creación de diseños

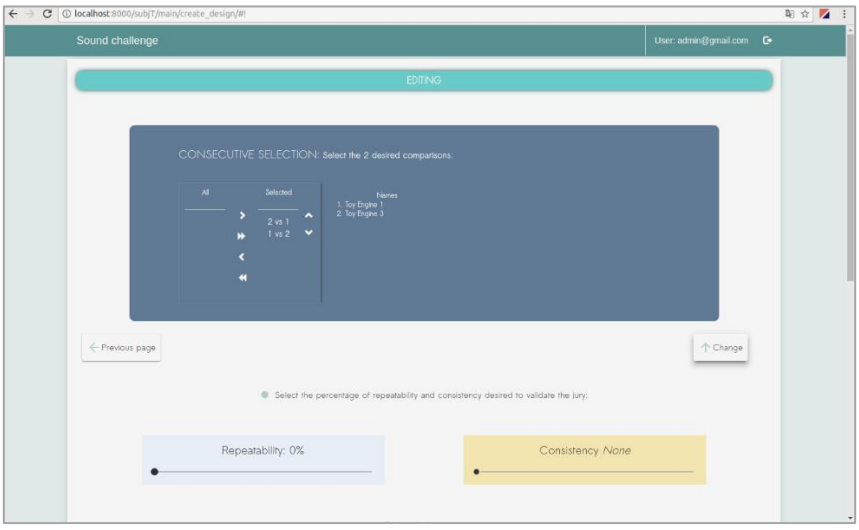
Si resulta que el diseño es uno ya existente con dos audios, se obtendría las siguientes vistas dependiendo de la opción de configuración de pares que se quiera utilizar:



(1)



(2)



(3)

Fig. 47: Vista de creación - (1) Selección de las dos opciones de diseño; (2) Opción de diseño de rejilla; (3) Opción de diseño consecutiva

Cumpliendo los requisitos, aparte de poder seleccionar los pares que se quieran y el orden de aparición que tendrán en el test, se define también la selección de los parámetros subjetivos a evaluar junto a los valores de repetitividad (valor umbral) y consistencia de umbral (repetitividad) que se auto calculan dependiendo de los pares que se seleccionen mostrado en las figuras 47 y 48.

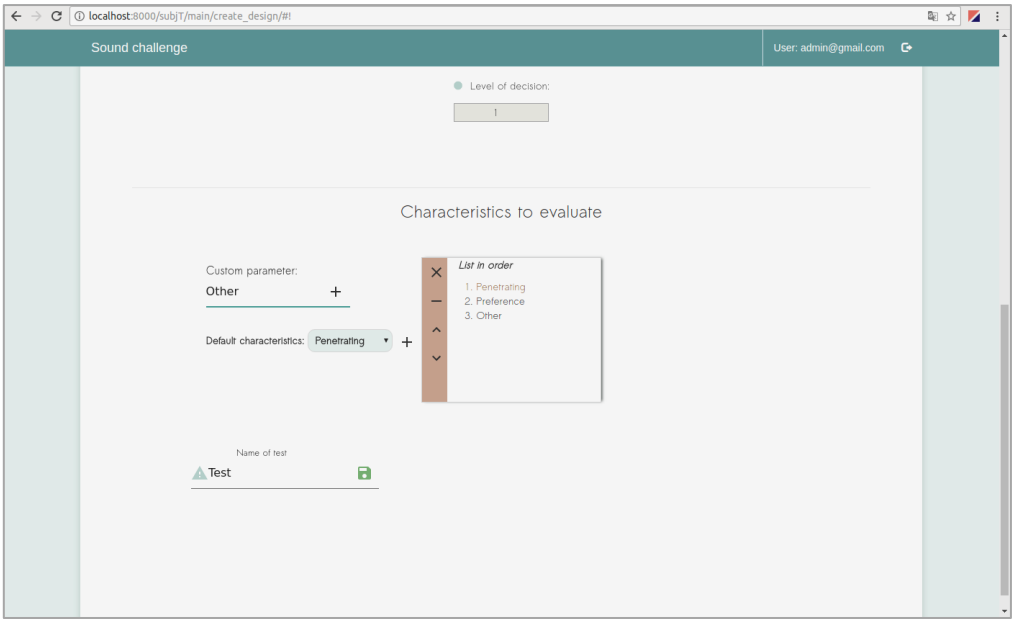


Fig. 48: Configuración de los parámetros subjetivos

Una vez se tienen todas las características del test configurado, el sistema comprueba mediante JavaScript que todo está correcto y procede a enviar los datos al controlador para que lo guarde en la tabla de diseños de la base de datos.

```
name_design = form['name_design']
sound_data = form['sound_data']
list_of_sounds = form['list_of_sounds']
par_data = form['par_data']
username = form['username']
email = form['email']
editing = form['isediting']
obj = Design()

same_name = Design.objects.filter(name=form['name_design'])
if same_name.exists():
    same_name.delete()

obj.name = name_design
obj.list_of_sounds = list_of_sounds
obj.sound_data = sound_data
obj.par_data = par_data
obj.editing = editing
obj.save()

return render(request,'subjT/main.html',{
    'if_admin': 'Yes', 'username': username,
    'email': email,
})
```

Fig. 49: Código de python del controlaador de creación de tests

Una vez guardado el diseño la aplicación redirigirá de nuevo a la vista principal de la aplicación además que aparecerá en la lista de pruebas a realizar.

5.2.2.5. Vista de inicio y configuración del test – *test_template.html*

Una vez se tiene el nuevo test diseñado, el siguiente paso es llevarlo a cabo. Los usuarios con permisos de administrador también pueden hacer las pruebas como si de un usuario normal se tratara. Para acceder a ellos hay que seleccionar la opción de *Listening tests* del navegador en la vista principal y aparecerá información sobre la prueba y la lista.

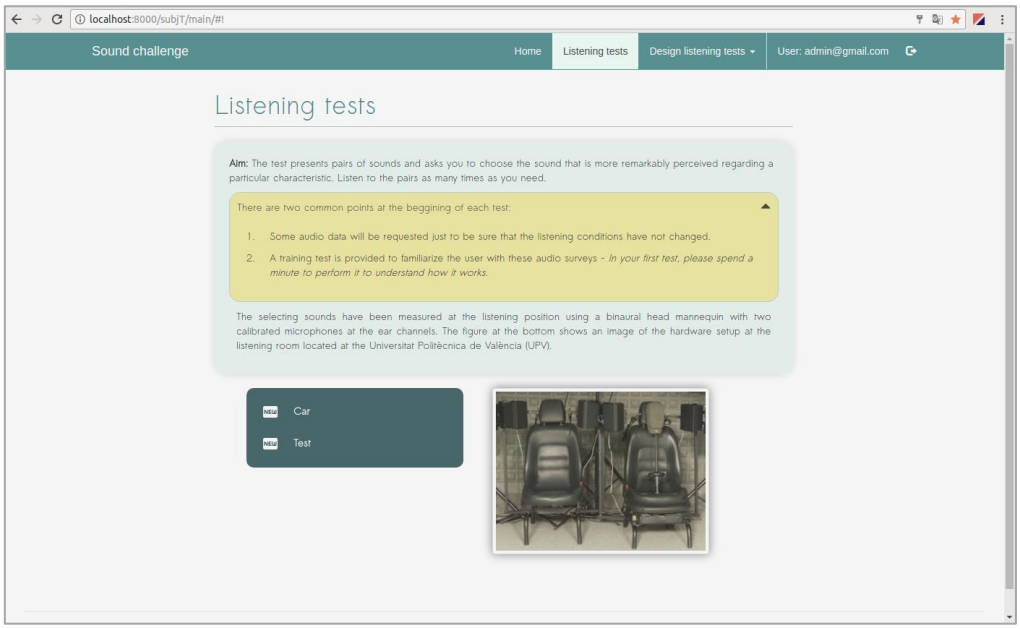


Fig. 50: Selección de pruebas de escucha formato PC - Vista principal

Cada vez que se accede a dicha pestaña igualmente que con los audios, se realiza una consulta asíncrona a la base de datos para acceder al modelo de los diseños y así listarlos, además de comprobar mediante el modelo de *UsersInTest* si el usuario en cuestión ha participado ya en los tests de la base de datos. Si no lo ha hecho, aparece un icono indicando que es un test 'nuevo', y si ya lo ha realizado, sale un símbolo que confirma que ya está hecho. Dicha consulta se realiza mediante AJAX, siendo ésta otra parte del controlador que está en la parte del cliente.

```
// Function to get all the designs on the BD:
function get_tests(callback){
    //Consulta con AJAX
    var tests ='';
    var users_in_test ='';
    $.ajax({
        url: "{% url 'subjT:get_data_test' %}",
        type:'get',
        success: function (response) {
            //lo que haces si es exitoso
            var aux = response['ser_tests'];
            var aux_1 = response['ser_users'];
            tests = JSON.parse(aux.replace(/&quot;/g, '"'));
            users_in_test = JSON.parse(aux_1.replace(/&quot;/g, '"'));

            callback(tests,users_in_test)
        }
    });

}
```

Fig. 51: Consulta AJAX de los tests y usuarios en el test - parte del controlador

Una vez se selecciona un test disponible, se envía un formulario para acceder a la vista de configuración de usuario en el test mostrado en la figura 53. El código del controlador que gestiona esta tarea se encuentra en la figura 52.

```
def conf_test(request,test_name,name):
    # signals = AudioFile.objects.all()
    all_test = Design.objects.all()

    if request.method == 'POST':
        form = request.POST
        for test in all_test:

            # The test exists on base data
            if form['name_test'] == test.name and test.editing == 'NO':
                username = form['username']
                email = form['email']
                test_name = form['name_test']

                return render(request, 'subjT/test_template.html', {
                    'if_admin': form['adm'],
                    'username': username,
                    'email': email,
                    'name_test': test_name,
                    })

        return render(request, 'subjT/main.html', {
            'if_admin': form['adm'], 'username': form['username'],
            'email': form['email'],
        })
    else:
        return HttpResponseRedirect('/subjT')
```

Fig. 52: Código python del controlador que enlaza a la vista de configuración del test

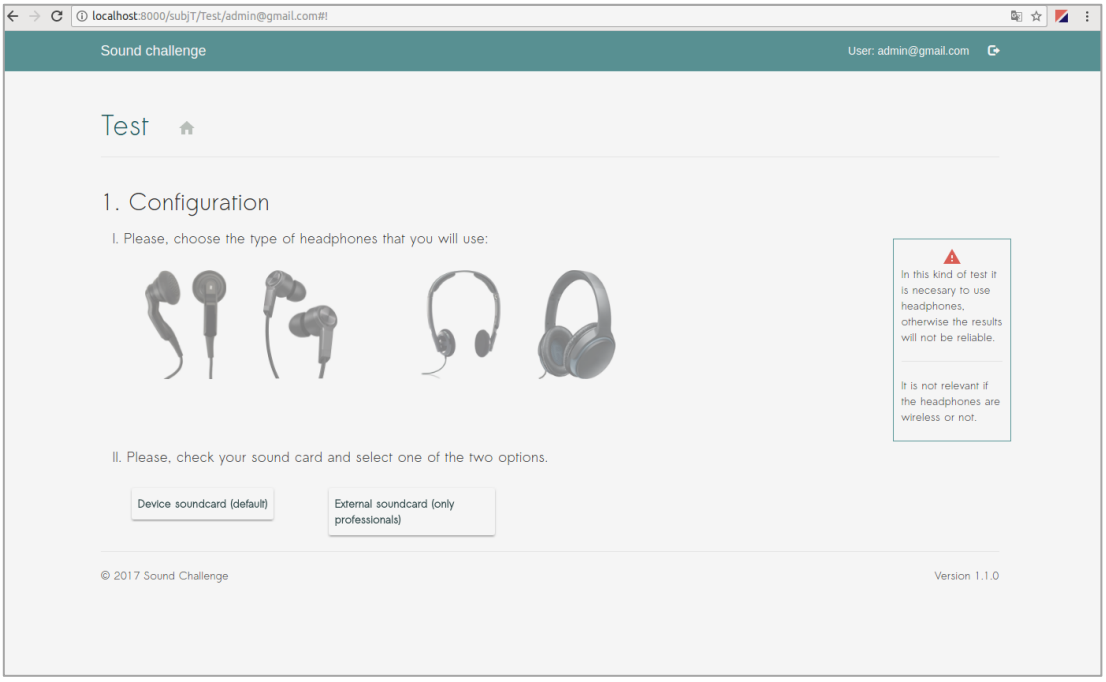


Fig. 53: Vista de configuración del test de usuario - 1a parte

Con esta vista se pretende cumplir el requisito de obtener más datos sobre la situación del usuario al realizar el test configurando qué auriculares va a llevar y la tarjeta de sonido que va a utilizar (la del dispositivo o una externa). Esta es la primera parte previa a la realización del test, cuando se ha seleccionado correctamente ambos requisitos, se procede a la segunda parte donde

se explican las instrucciones de la prueba y un entrenamiento para que el usuario se habitúe a cómo funciona la prueba (figura 54).

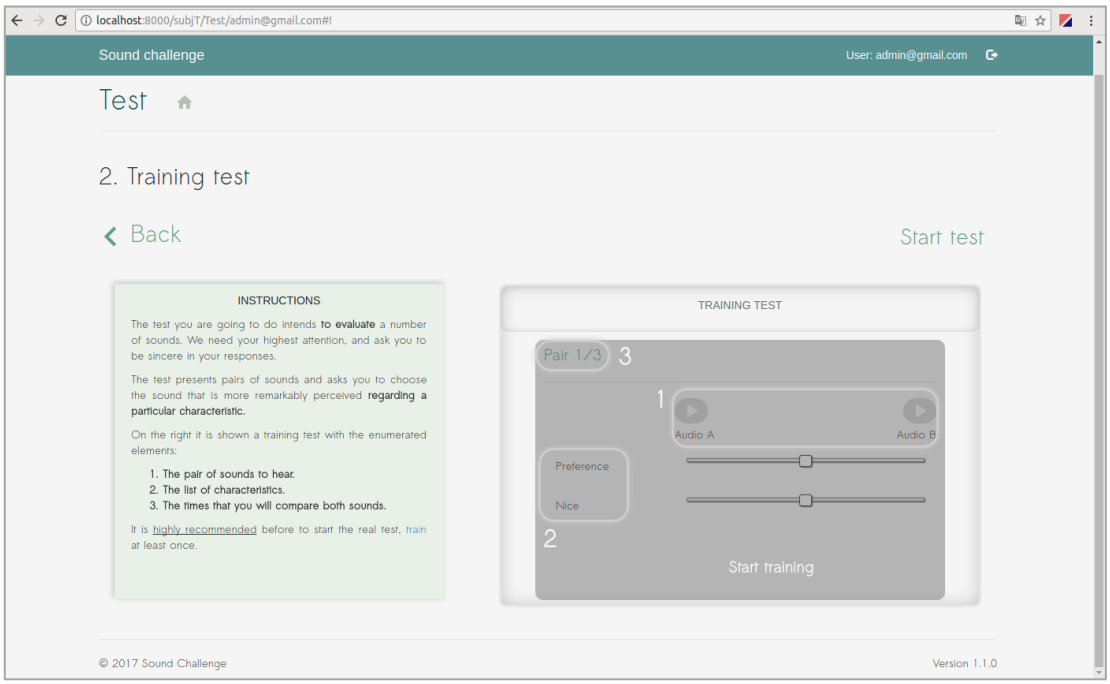


Fig. 54: Vista de configuración del test de usuario - 2a parte

Cuando el usuario desee comenzar el test, pulsará en el botón superior *Start test*. En ese momento se enviarán los datos de configuración introducidos al controlador que enlaza esta vista con la siguiente (figura 55).

```
def StartTest(request,test_name,name):
    if request.method == 'POST':
        form = request.POST
        username = form['username']
        email = form['email']
        test_name = form['name_test']
        headphone = form['headphone']
        sound_card = form['sound_card']
        all_test = Design.objects.all()

        return render(request, 'subjT/running_test.html', {
            'if_admin': form['adm'],
            'list_tests': all_test,
            'username': username,
            'email': email,
            'name_test': test_name,
            'headphone': headphone,
            'sound_card': sound_card,
        })
    else:

        return HttpResponseRedirect('/subjT')
```

Fig. 55: Código python definiendo la función de controlador de la vista de configuración de test

5.2.2.6. Vista de ejecución del test – *running_test.html*

Vista que se configura con las opciones guardadas en el diseño. Es la prueba de escucha en sí, la longitud de ésta depende del número de pares que se hayan configurado y de cuantos elementos subjetivos tenga que evaluar el usuario. Está programado para que cada pantalla de cada par sea ligeramente distinta con la anterior pero sin cambiar la estructura total. Dichos cambios dinámicos están programados con JavaScirpt. A continuación se muestra el primer par y el último del test de prueba.

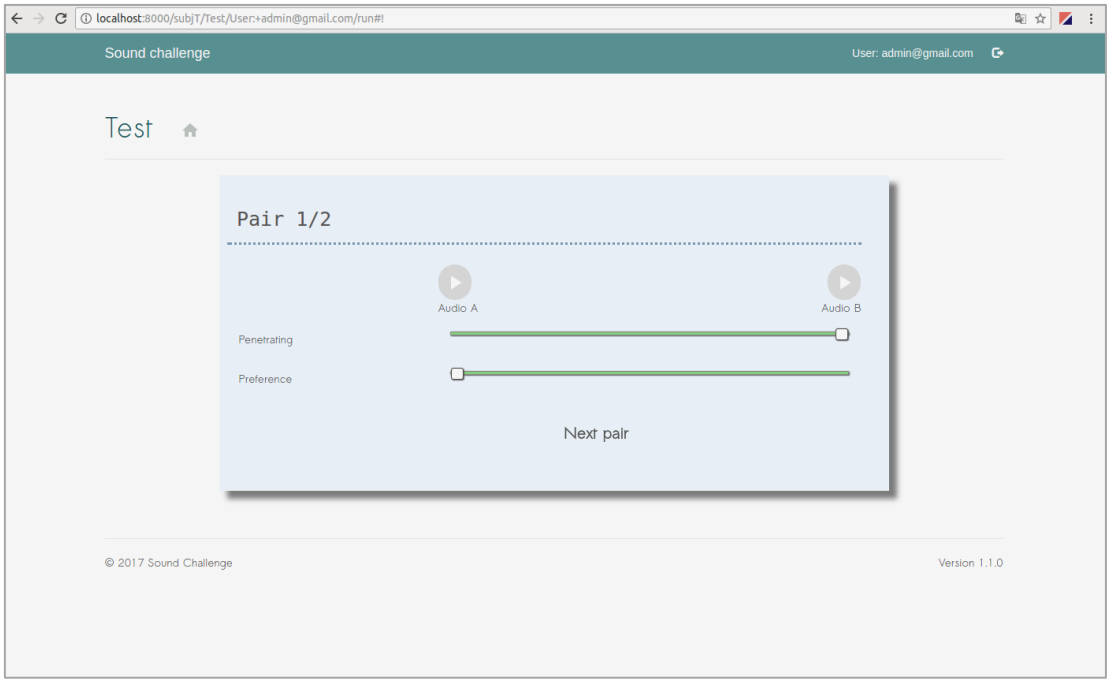


Fig. 56: Vista de ejecución del test - Primer par del test de prueba

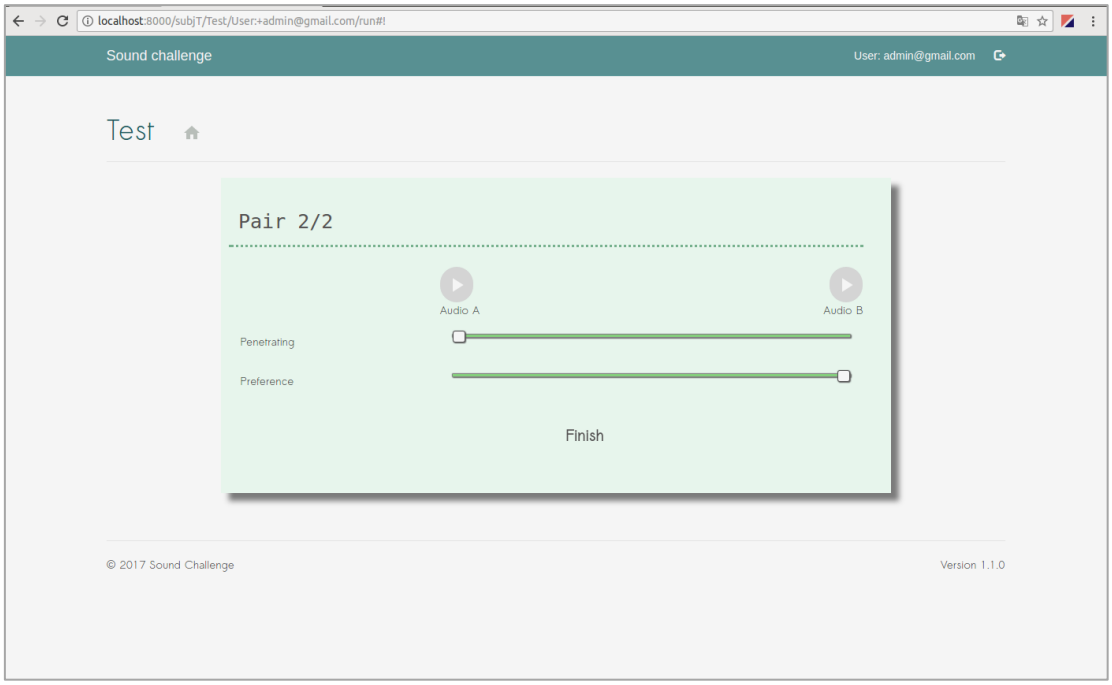


Fig. 57: Vista de ejecución del test - Segundo y último par del test de prueba

En cualquier momento se puede salir del test clicando en el icono de una casa de arriba pero no se guardarán las respuestas y la configuración en la base de datos, a no ser que se pulse en *Finish.* En ese caso los datos se envían mediante un formulario al controlador que guarda los datos en la tabla de *UserInTests* y además enlaza con la última vista implementada, la de dar las gracias al usuario y dar un pequeño *feedback* inmediato. Dicho código de controlador se muestra en la figura 58. Uno de los datos guardados aparte de las elecciones y configuración, es el valor de repetitividad y consistencia que ha conseguido el usuario para su posterior análisis.

```
def Message(request):

    if request.POST:
        form = request.POST
        Design.objects.all().as_manager()
        design = get_object_or_404(Design, name=form['name_test'])
        all_users = design.usersintest_set.all()
        user_exist = all_users.filter(email=form['email'])
        user_exist.delete()
        design.usersintest_set.create(email=form['email'],
                                      data=form['choice_data'])

        return render(request,'subjT/thanks.html',{
            'username': form['username'],
            'if_admin': form['adm'],
            'email': form['email'],
            'name_test': form['name_test'],
        })
    else:
        return HttpResponseRedirect('/subjT')
```

Fig. 58: Código python del controlador entre la vista de ejecución del test a la del mensaje

5.2.2.7. Vista de finalización del test – *thanks.html*

Una vez el usuario finaliza el test, se le redirige a esta vista donde se le da las gracias por ayudar a la investigación y se le muestra, si no es el primero, una gráfica orientativa donde se le muestra su audio más preferido dentro del resto de usuarios que han realizado el test.

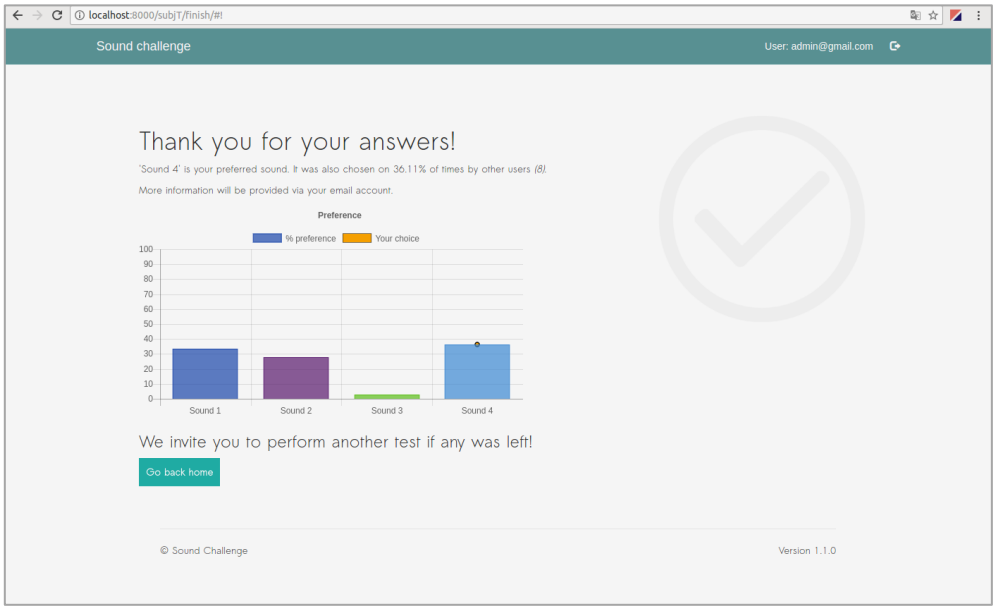


Fig. 59: Vista del feedback simple sobre el test terminado

Una vez se clica el botón de regresar al *home*, el controlador simplemente enlaza esta vista con la principal mostrando como opción seleccionada el *home*. El código implementado en este caso es el siguiente:

```
return render(request, 'subjT/main.html', {
    'if_admin': form['adm'], 'username': form['username'] ,
    'email': form['email'],
})
```

Fig. 60: Código python del controlador para retornar a la página principal

5.2.2.8. Visualización de resultados – opción análisis

Por último una de las especificaciones impuestas al proyecto es de tener integrado el análisis de las pruebas de escucha dentro de la aplicación. Este último apartado del diseño muestra el sistema de análisis definido dentro de la vista principal de la aplicación.

Para acceder a ella hay que ser administrador accediendo desde el navegador superior de *Sound Challenge* mediante la misma opción que permitía acceder a la creación de diseños (*Design listening test*, figura 43). La segunda opción es la del análisis. La página principal en esta opción se visualiza de la siguiente manera:

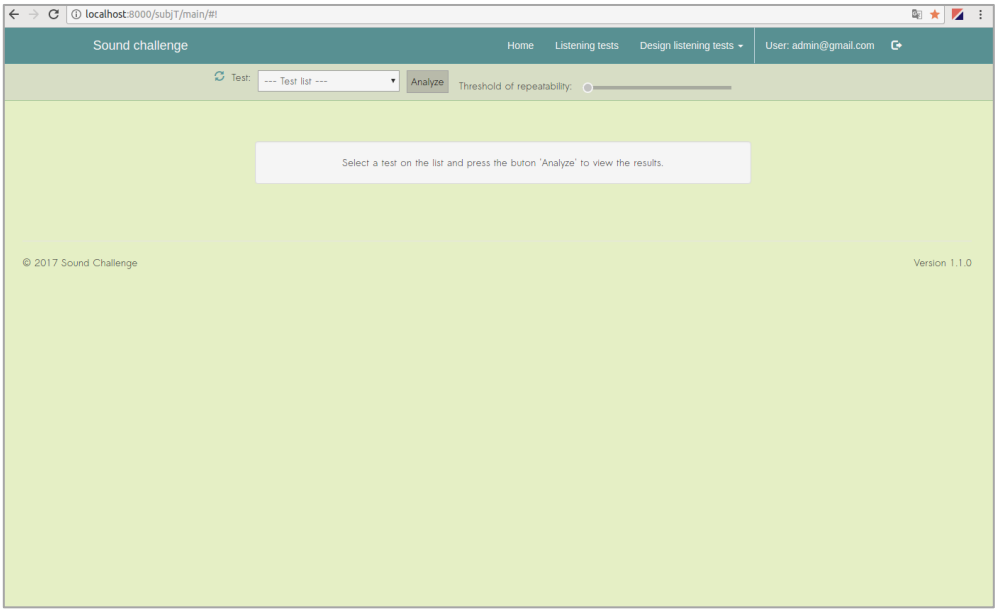


Fig. 61: Vista de análisis de pruebas

Una vez se selecciona un test de la lista cargado igualmente con una petición AJAX mostrada anteriormente, se generan los elementos necesarios para mostrar los datos y crear gráficas. Se realizan una serie de cálculos estadísticos contando los usuarios en el test que sean válidos, decidido por el valor de repetitividad.

Los valores se calculan y grafican dependiendo del valor definido en la creación del diseño, pero como se puede observar en la figura 61, en la parte superior hay un *slider* el cual permite seleccionar el umbral de repetitividad que se desea para mostrar los datos.

Todo el código necesario para realizar el análisis se encuentra en el anexo 2. Como se puede comprobar, está totalmente implementado en JavaScript, cada vez que se presiona el botón *analyze* se ejecuta todo ese código. En concreto este código es una migración del código

implementado en la aplicación de tests subjetivos de Matlab comentado al principio del presente documento. La visualización de los posibles resultados se puede ver en el siguiente punto.

5.3. Resultados de *Sound Challenge*

A continuación se muestran los resultados obtenidos en la aplicación web de *Sound Challenge* siguiendo un ejemplo denominado '*Car*'. Con ello se demuestra el potencial y alcance que posee dicha aplicación para evaluar los distintos sonidos.

La aplicación, como se ha visto anteriormente, muestra los datos de análisis de cada uno de las pruebas realizadas por los usuarios. Por comodidad a la hora de mostrar los datos se va a mostrar la vista del análisis de resultados en formato de móvil.

El test de ejemplo '*Car*' posee estas características:

- Número de audios: 4
- Número de pares implementados: 9
- Parámetros subjetivos a evaluar: Preferencia y molestia.

Ya que consta de 9 pares, se decidió que el valor umbral de repetitividad especificado en el diseño fuera del 66% donde de 3 pares que se deban cumplir la repetitividad, el usuario podría fallar uno pero los otros dos los elegiría a conciencia. Con todo ello los resultados hallados gracias a 10 participantes en la prueba fueron los siguientes.

Número de usuario	Consistencia	Repetitividad
1	100.00%	100.00%
2	100.00%	66.67%
3	100.00%	100.00%
4	100.00%	100.00%
5	100.00%	100.00%
6	100.00%	100.00%
7	50.00%	100.00%
8	100.00%	100.00%
9	50.00%	100.00%

Tabla 5: Consistencia y repetitividad de los usuarios en 'Car'

Resulta que de 10 participantes solo han pasado el umbral de 66% de repetitividad 9. Estos mismos datos y el valor de concordancia entre los usuarios con respecto a los parámetros se muestran de la misma manera en la aplicación (figura 62). Como se puede observar en ambos parámetros se tiene la misma concordancia, esto significa que en un par si prefieren uno de los dos audios el otro lo etiquetan como molesto.

Los valores desvelan que el 50% de los usuarios están de acuerdo con respecto a los parámetros para cada audio.

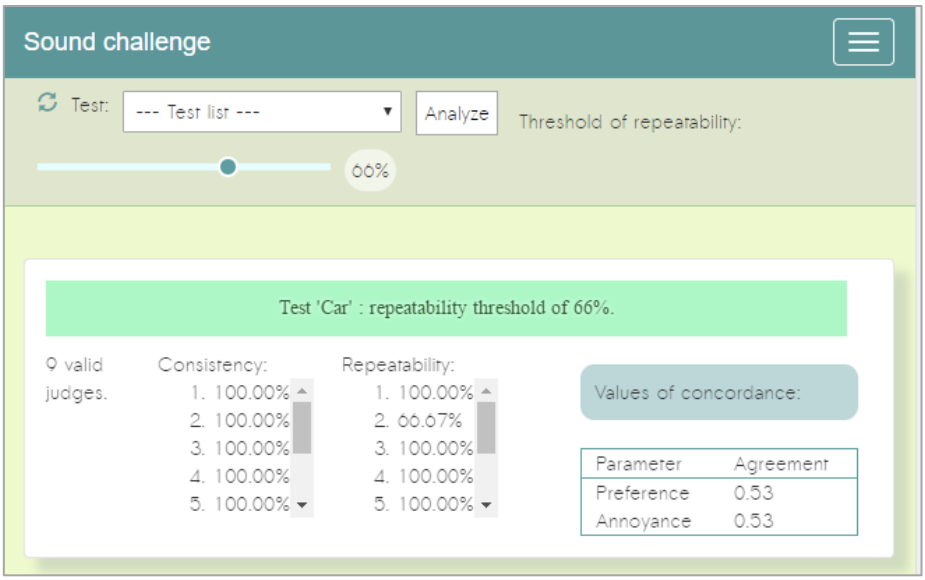


Fig. 62: Resultados del test 'Car' en la aplicación – 1ª parte

Los valores que quedan por analizar son la probabilidad de preferencia de los audios y los valores de mérito de los parámetros en cada audio. Estos datos se muestran en la aplicación de forma muy representativa y clara como se puede comprobar en la figura 63.

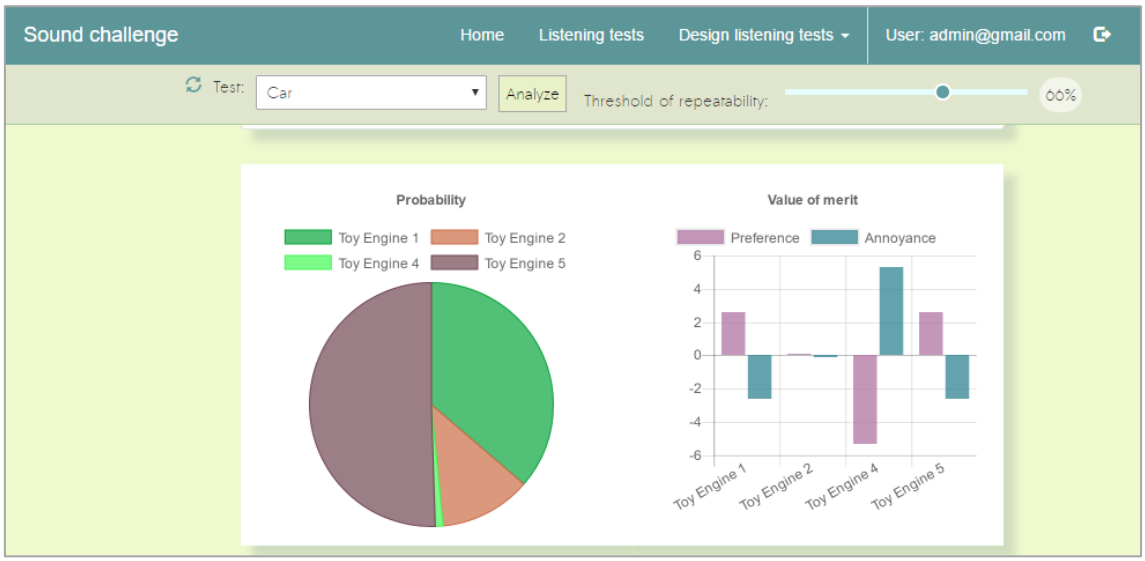


Fig. 63: Valores de probabilidad y valores de mérito de 'Car'

Se puede observar que el sonido más preferido es el denominado *Toy engine 5* y el que menos el *Toy Engine 4* siendo concordante con los valores obtenidos en la gráfica de valores de mérito, donde *Toy Engine 4* se considera el más molesto y el *Toy Engine 5* junto al *Toy Engine 1* son los menos molestos y más preferidos.

Con solo 10 usuarios se pueden sacar conclusiones con respecto a las características de los audios, gracias al diseño de las pruebas y todo su cálculo estadístico realizado en el análisis.

Estos resultados se han hallado con el objetivo de corroborar que la aplicación funciona correctamente, ya que es una prueba de escucha sobre sonidos de motores de juguetes que fueron

recuperados del estudio realizado con niños en [7]. Con ello se ha demostrado que el sistema de creación de pruebas, su configuración y ejecución está hecho de manera que se adapte a cualquier situación con respecto a los audios y así resultar una herramienta versátil y flexible para la evaluación de audio subjetivo.

Capítulo 6. Conclusiones y líneas futuras

Finalmente, en el presente capítulo se recopilan las conclusiones obtenidas basándose en los objetivos principales.

Además, al ser un trabajo realizado con la intención de utilizarse como herramienta online, se exponen diferentes opciones para la mejora y optimización de la aplicación, las cuales serían las líneas futuras del proyecto.

Capítulo 6. Conclusiones y líneas futuras

6.1. *Conclusiones*

A lo largo de este trabajo fin de máster se han podido ver diferentes conceptos, tanto teóricos en materia de evaluación de las características subjetivas del sonido como más técnicos sobre programación. La tarea de aunar ambos temas en un solo proyecto y documentarlo bien fue complicado, pero no imposible. Se ha podido conseguir hacer una herramienta bastante completa y sólida que permite realizar evaluaciones que normalmente se deben hacer de forma local.

Cabe decir que la aplicación aún sigue en desarrollo porque el objetivo es evaluar los sonidos procedentes del sistema de ecualización de cancelación de sonido multiusuario y a día de hoy, dicho sistema va a la par en términos de estudio y desarrollo. Por otro lado, viendo cómo resultaba la aplicación han ido surgiendo ideas nuevas que se quieren implementar y por ello aún queda trabajo que hacer.

Este trabajo ha sido una oportunidad para aprender de forma masiva conceptos de programación web además de conocer de la existencia del *framework* Django y de su funcionamiento. El haber desarrollado todo el código de forma individual me ha permitido abrirme muchas puertas en desarrollo. Pero no todo ha sido sencillo, la transición de no conocer Django a realizar una aplicación completa fue difícil y hubo muchos errores al principio, pero con paciencia y tiempo se pudieron resolver y llegar a crear lo que ahora es *Sound Challenge*. Ayudó que tenía conocimientos básicos sobre HTML, CSS y JS. De forma autodidacta he podido aprender enormemente muchas más técnicas y procesos que antes no sabía.

Destacar que la mayoría del tiempo ha sido invertido en la creación del código, sobretodo en la programación de JavaScript y CSS, intentando crear una interfaz gráfica agradable y sencilla. Al principio del desarrollo de la interfaz no se tuvo en cuenta realizarlo en formato móvil, por ello una de las semanas más importantes e intensas fue la inclusión del *framework* Bootstrap ya que sin saber cómo funcionaba al principio, se consiguió cambiar toda la estructura de la aplicación web mediante las filas y columnas que Bootstrap utiliza.

Desde una perspectiva técnica se puede confirmar que la mayoría de objetivos han sido cumplidos siendo la excepción la no evaluación de los sonidos del sistema de ecualización, pero la aplicación ya está preparada para poder cargar los audios y realizar diferentes pruebas. Aportando finalmente valor al proyecto *D-NOISE* por el cual se decidió realizar este trabajo.

6.2. Líneas futuras

Como se ha comentado varias veces, *Sound Challenge* aún sigue en proceso de desarrollo, por ello aquellas mejoras o extras que se quieren añadir a la aplicación son las explicadas a continuación como líneas futuras.

1. Mejoras según el diseño y evaluación de las pruebas subjetivas:

   - Hacer tests con personas expertas en el tema para obtener unos datos de referencia fidedignos y así evitar en todo lo posible errores.
   - Hacer un estudio para averiguar qué parámetros son los más adecuados a la hora de realizar evaluación subjetiva de los audios de un sistema de ecualización de cancelación de sonido multiusuario.

- Añadir una pequeña explicación a los parámetros subjetivos para intentar normalizar las elecciones de los jueces y así asegurar medianamente el entendimiento del test para el juez.

2. Mejoras en el software:

- Llevar la parte del cálculo estadístico del controlador al modelo para que no exista ningún procesado de datos en el cliente y así asegurar que cuando existan cientos de usuarios, no se colapse. Además, si se analiza desde varios puntos de la web, se reutiliza dicho código.
- Permitir la previsualización de los resultados de los tests a todos aquellos usuarios que hayan hecho el test, para que tengan un *feedback* inmediato dentro de la aplicación sin tener la necesidad de enviar un correo con los resultados a cada usuario.
- Realizar la verificación del correo electrónico para permitir la recuperación de la cuenta por si se olvida la contraseña, utilizando por tanto el e-mail para dicho cometido.
- Implementar un sistema de descarga de las gráficas y datos visuales así como crear un archivo *.csv* con todos los datos pertenecientes del test.
- Añadir una opción de filtrado en el análisis donde se utilice la información sobre el género y la edad de los usuarios para tener un sesgo mayor a la hora de analizar.
- Añadir una opción de filtrado en la selección de audios dependiendo de las características introducidas en los campos del modelo.

Bibliografía

[1] iTEAM - UPV, «Instituto de Telecomunicaciones y Aplicaciones Multimedia (iTEAM),» [En línea]. Available: http://www.iteam.upv.es/waves.php.

[2] A. Gonzalez, M. de Diego, M. Ferrer y G. Piñero, «Multichannel Active Noise Equalization of Interior Noise,» *IEEE Transactions on audio, speech and language processing,*, vol. 14, nº 1, pp. 110-122, January 2006.

[3] A. Gonzalez, M. Ferrer, M. de Diego, G. Piñero y J. Garcia-Bonito, «Sound quality of low-frequency and car engine noises after active noise control,» *Journal of Sound and Vibration*, vol. 265, nº 1, pp. 663-679, 2003.

[4] A. Gonzalez, M. Ferrer, M. de Diego y G. Piñero, «Subjective considerations in multichannel active noise control equalization of repetitive noise,» de *Active 2002*, Southampthon, UK, 2002.

[5] M. Cartwright, B. Pardo, G. J. Mysore y M. Hoffman, «Fast and easy crowdsourced perceptual audio evaluation,» de *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference*, 2016.

[6] N. C. Otto y G. H. Wakefield, «A Subjetive Evaluation and Analysis of Automotive Starter Sounds,» *Noise Control Engineering Journal*, vol. 41, nº 3, pp. 377-382, Noviembre-Diciembre 1993.

[7] G. Piñero, L. Fuster, A. Gonzalez, M. de Diego, M. Ferrer, K. Pernias y M. Romero, «Sound quality evaluation of powered riding toy noises by children,» de *Internoise 2012*, New York, 2012.

[8] A. Alos Moya, *Desarrollo de una aplicación Android para la gestión de bases de datos, AcidSQL*, Valencia: UPV, 2015.

[9] M. A. Alvarez, «DesarrolloWeb.com,» 2 Enero 2014. [En línea]. Available: https://desarrolloweb.com/articulos/que-es-mvc.html.

[10] Cake Software Foundation, Inc, «CakePHP,» [En línea]. Available: https://book.cakephp.org/2.0/es/cakephp-overview/understanding-model-view-controller.html.

[11] campusMVP, «campusMVP,» 25 Agosto 2015. [En línea]. Available: https://www.campusmvp.es/recursos/post/Desarrollador-web-Front-end-back-end-y-full-stack-Quien-es-quien.aspx.

[12] F. J. Martínez Zaldivar, *Integración de servicios telemáticos - Tema 1: Introducción a HTML5.*

[13] W3Schools, «W3Schools,» [En línea]. Available: https://www.w3schools.com/html/. [Último acceso: Junio 2017].

[14] W3schools, «W3schools,» [En línea]. Available: https://www.w3schools.com/css/. [Último acceso: Junio 2017].

[15] F. J. Martínez Zaldivar, *Integración de servicios telemáticos - Tema 5: Hojas de estilo en cascada: CSS3 (Cascading Style Sheets, versi´on 3).*

[16] W3school, «W3school,» [En línea]. Available: https://www.w3schools.com/css/css3_intro.asp.

[17] Polymer Proyect, «Polymer Project,» [En línea]. Available: https://www.polymer-project.org/. [Último acceso: Junio 2017].

[18] desarrolloweb.com, «desarrolloweb.com,» [En línea]. Available: https://desarrolloweb.com/javascript/#quees. [Último acceso: Junio 2017].

[19] W3school , «W3School Javascript,» [En línea]. Available: https://www.w3schools.com/Js/.

[20] Django Software Foundation , «django,» [En línea]. Available: https://www.djangoproject.com/. [Último acceso: Junio 2017].

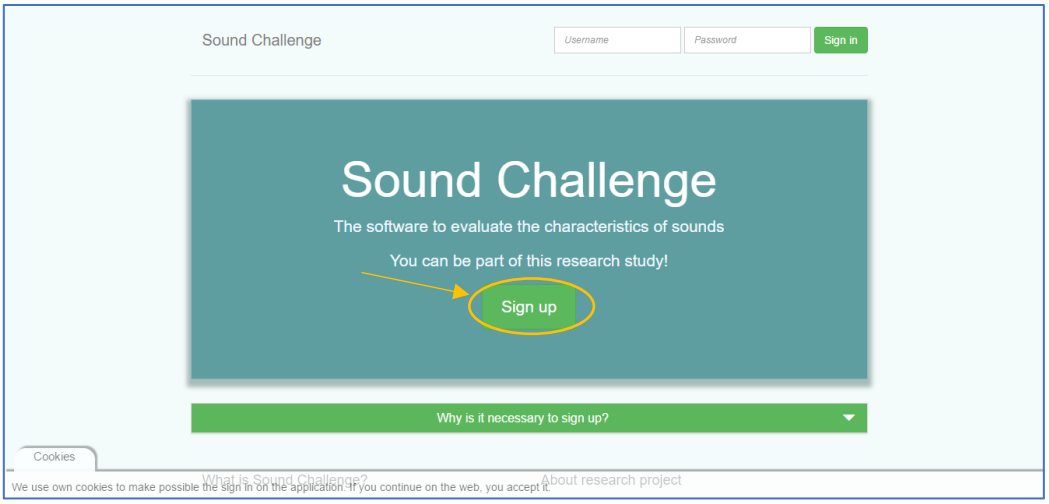
[21] Bootstrap Project, «Bootstrap,» [En línea]. Available: http://getbootstrap.com/.

ANEXO 1: Guía de usuario rápida

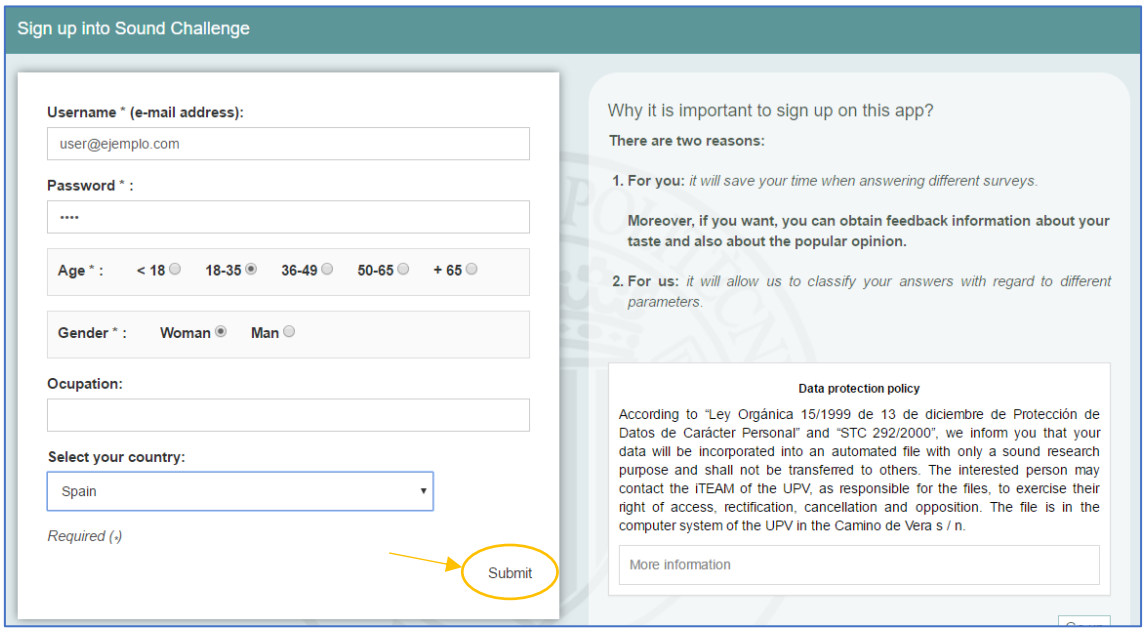
El presente anexo se ha pensado para servir de guía de usuario para aquellos que quieran utilizar la aplicación web en su totalidad.

Registro y realización de tests

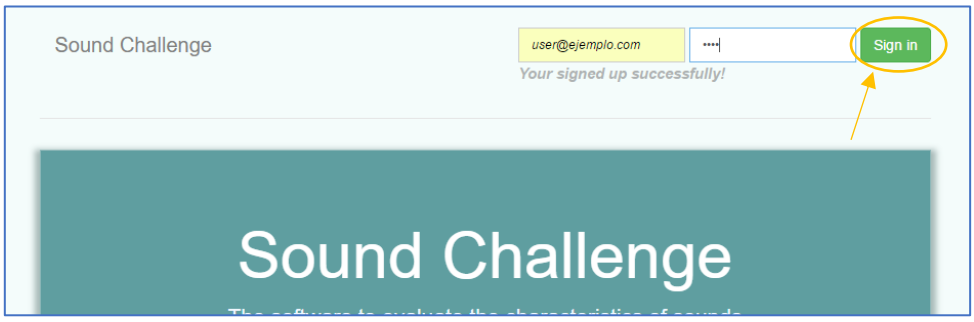
1. Registrarse en la página de inicio de *Sound Challenge* accediendo con el botón *Sign up*:



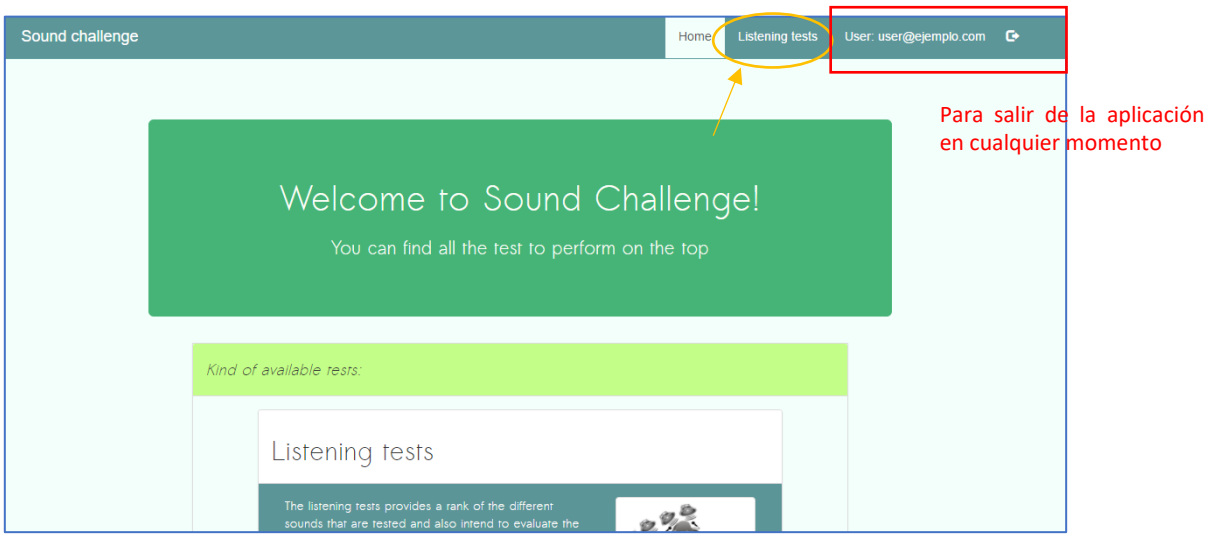
2. Aparecerá el formulario de registro. Rellenar de forma obligatoria el nombre de usuario con un e-mail, la contraseña, el género y la edad, los otros dos campos son opcionales. Una vez rellenado clicar el botón de *Submit*.



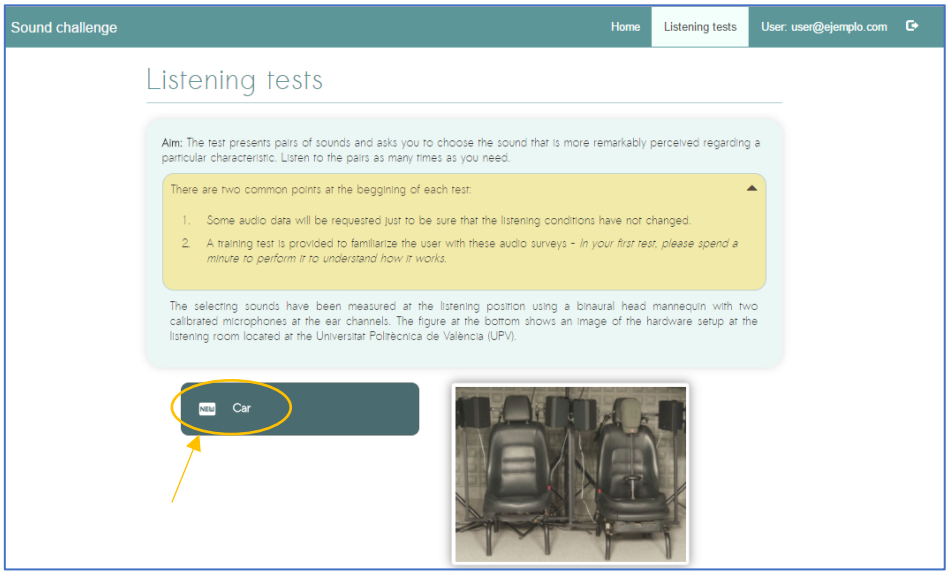
3. Acceder desde la página de inicio de *Sound Challenge* mediante el formulario ubicado en la parte superior dándole al botón de *Sign in*.



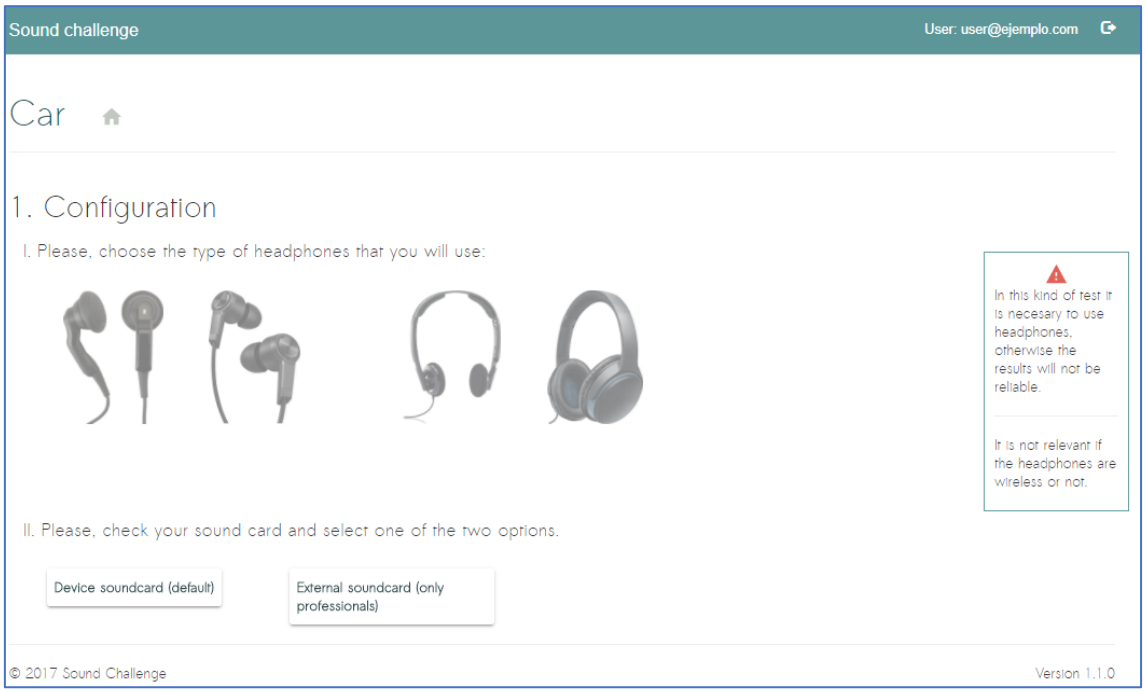
4. Aparecerá la pantalla de bienvenida nada más entrar, ahí hay información relevante a los test de escucha. Pero para acceder a las pruebas de escucha hay que clicar en la parte superior donde pone *Listening tests*.



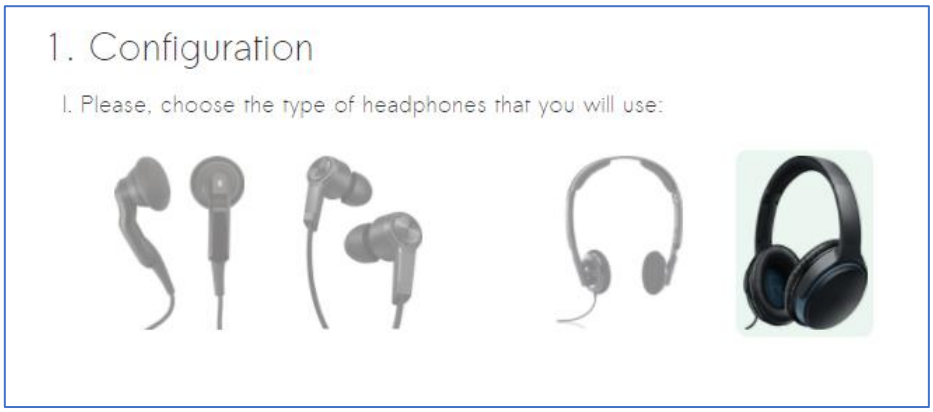
5. Al clicar en los tests aparecerá la lista y más información relevante a ellos. Seleccionamos un test de la lista.



6. Aparecerá la configuración nuestra para realizar el test. Es importante dedicar un minuto a la introducción de los datos de forma correcta.

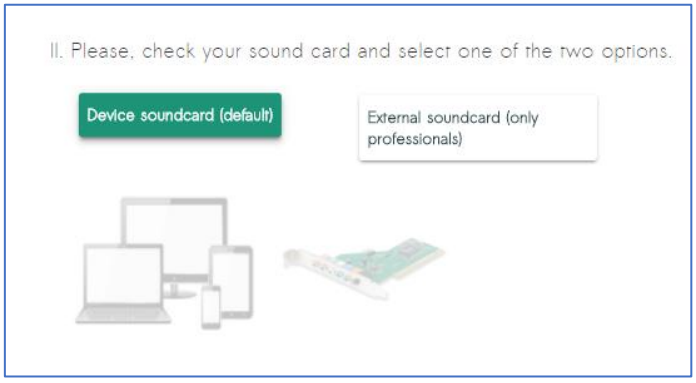


Configurar los auriculares:

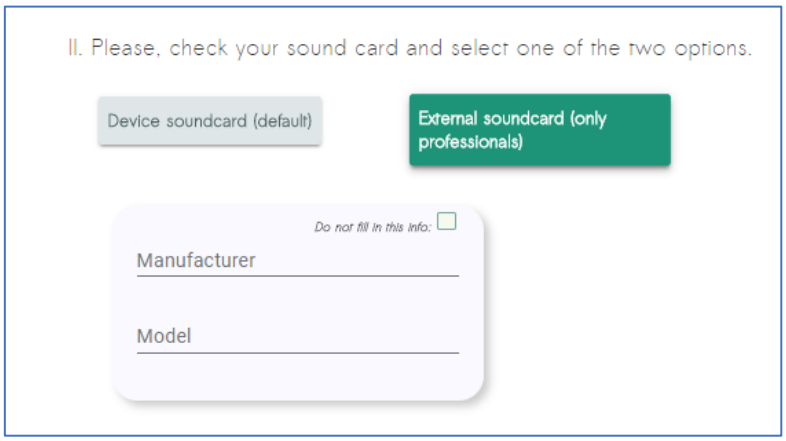


Configurar la tarjeta de audio:

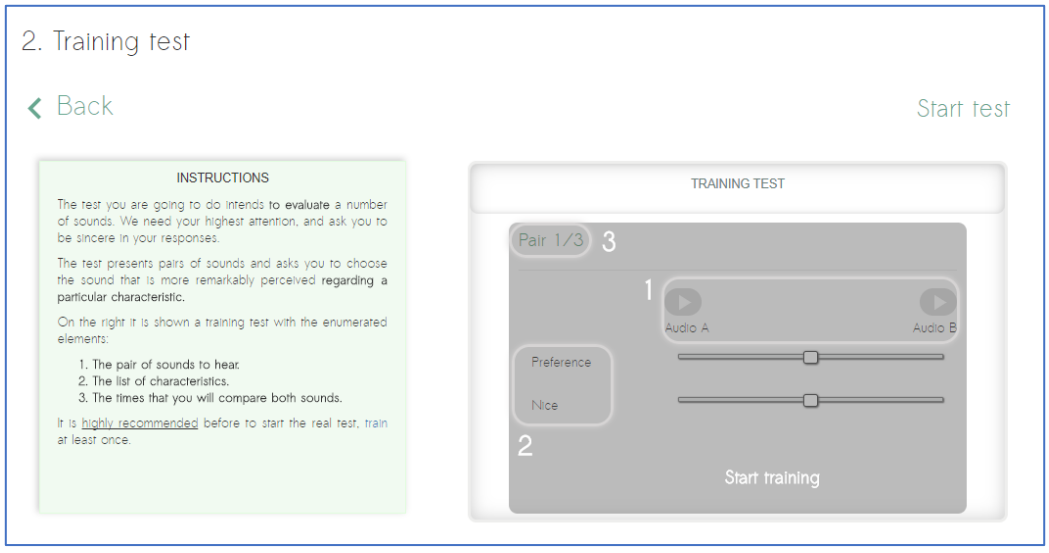
- Tarjeta del dispositivo: puede ser la del mismo PC, portátil, Tablet o móvil, o una tarjeta de audio específica que se ha añadido a la torre del PC (PCI):



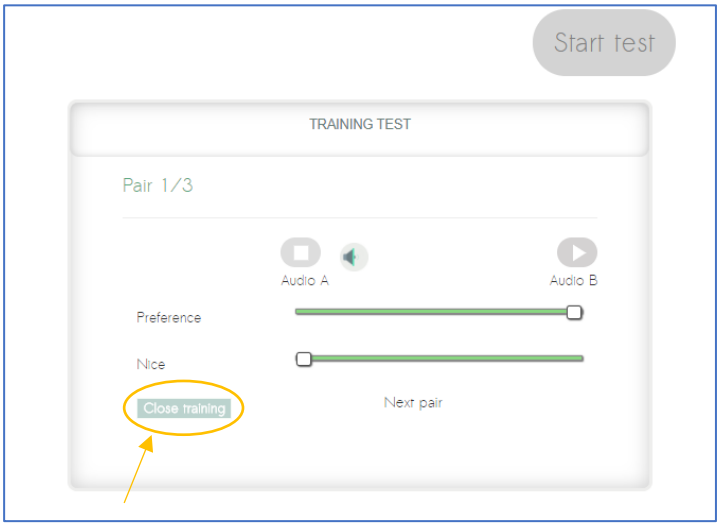
- Tarjeta de audio externa: Normalmente son los profesionales quien tienen una. Hay que especificar fabricante y modelo o dejarlo vacío.



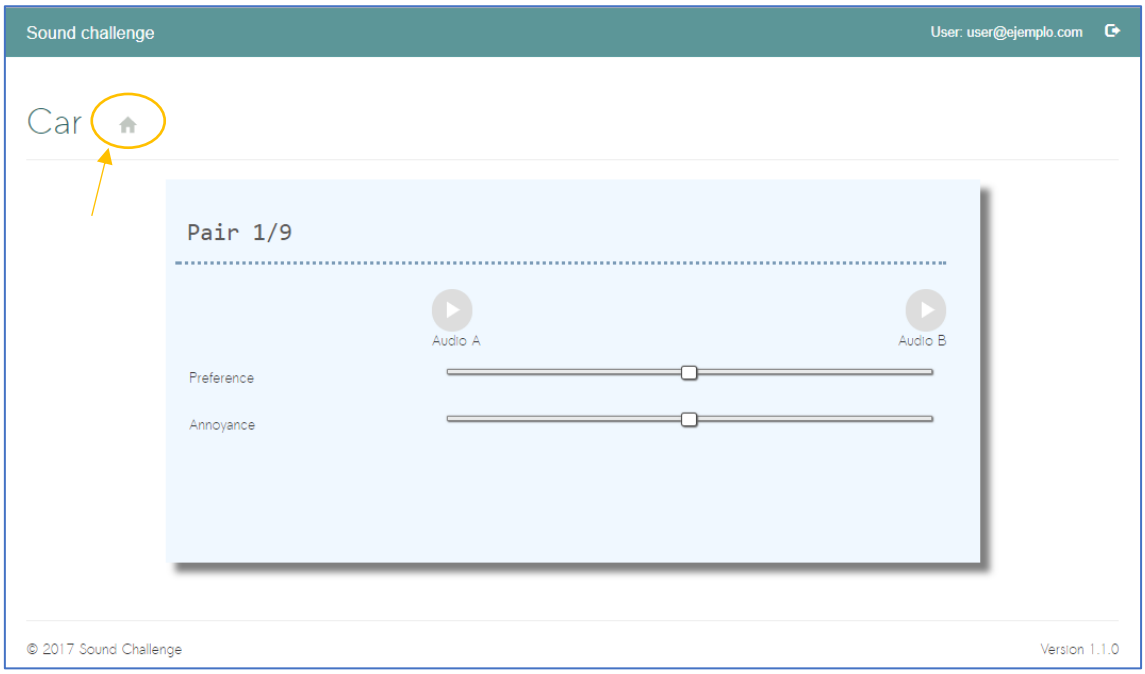
7. Cuando esté todo configurado correctamente aparecerá un botón de siguiente paso, le clicamos y nos pasa a la siguiente ventana de la zona previa del test, el entrenamiento. Esta parte está pensada para que entiendas cómo funciona la prueba y puedas practicar con unos sonidos de ejemplo. Es realmente importante realizarla al menos una vez.



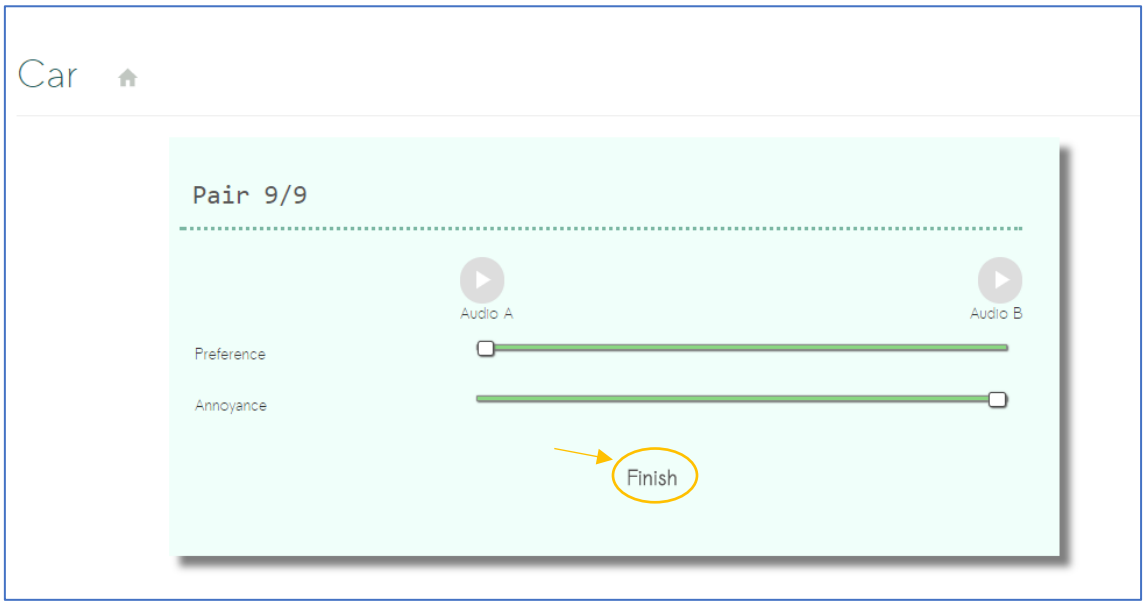
Mientras estés realizando el test de entrenamiento, no se puede empezar el test real, hay que terminarlo o cerrarlo con *close training.*



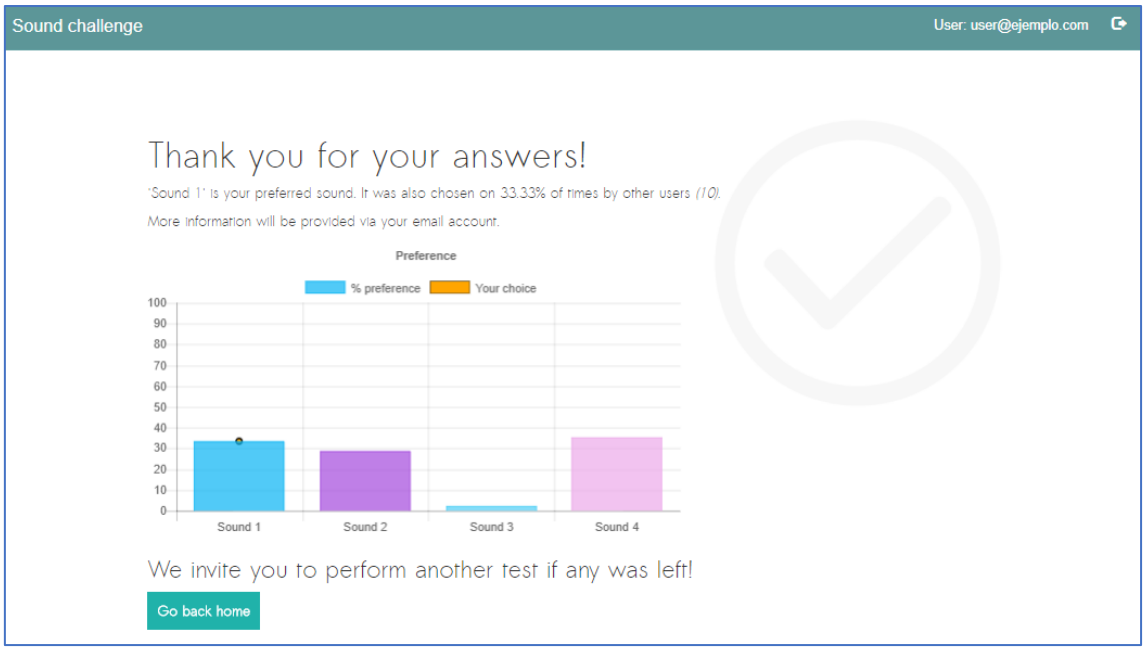
8. En la realización del test aparecerá esta pantalla mostrando el número de pares a evaluar en la parte de arriba de la hoja del test. Se puede salir en cualquier momento clicando en la casa que aparece al lado del nombre del test. Decide bien ya que no se puede ir a un par anterior.



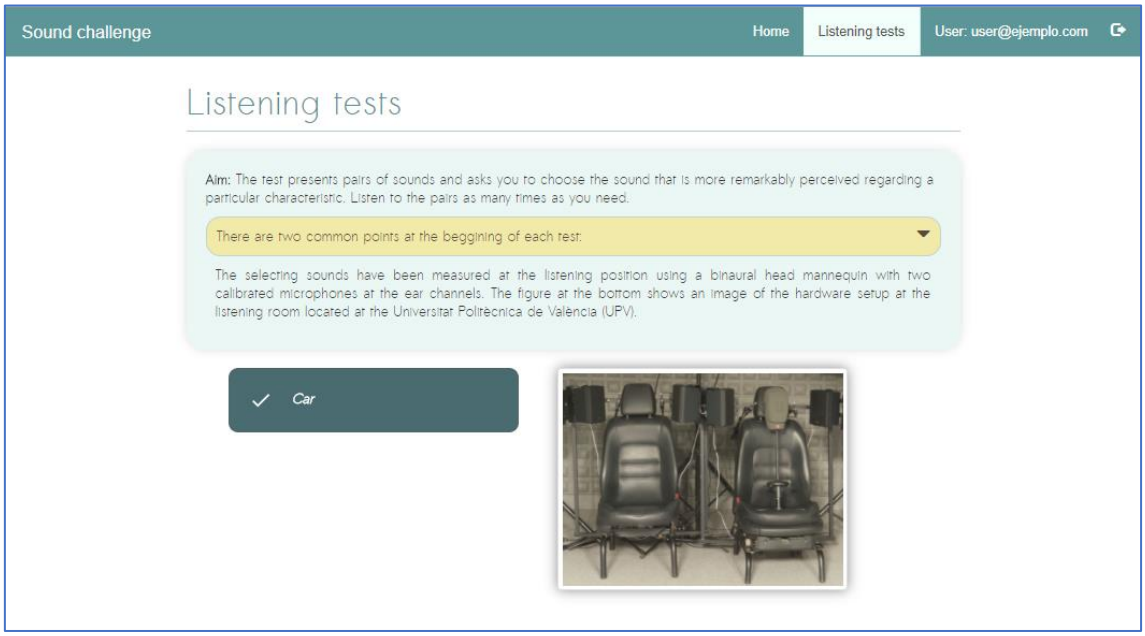
9. En el último par saldrá el botón *finish* para finalizar el test:



10. Una vez se termina, aparece una pantalla parecida a la siguiente donde te da una estimación de tus respuestas respecto a los demás usuarios que han hecho el test:



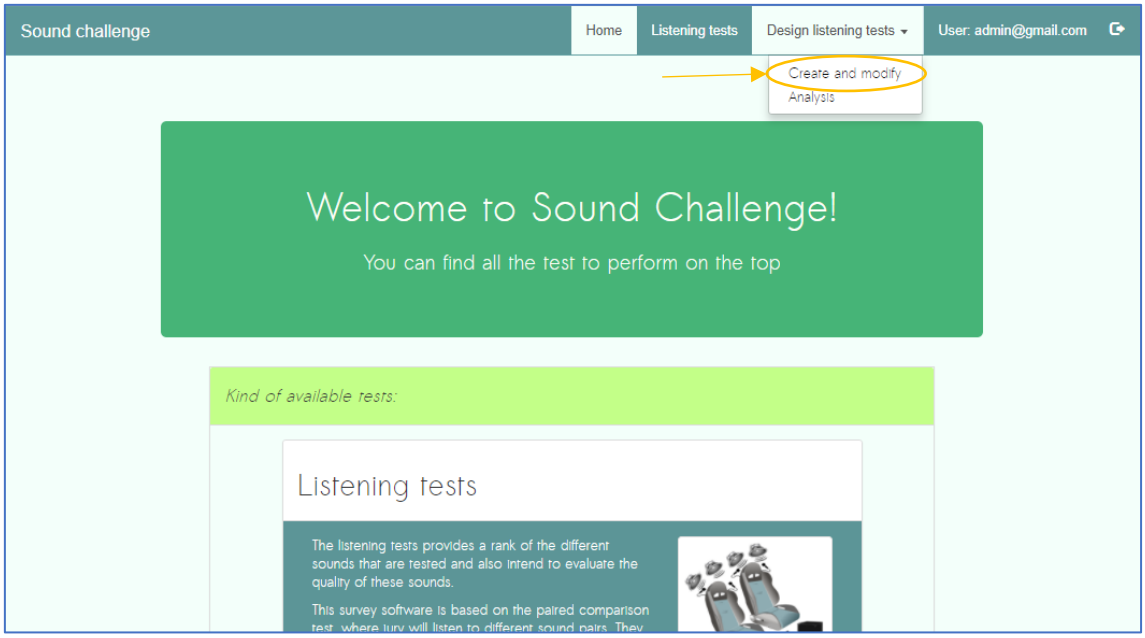
Para volver a la pantalla principal simplemente clicar en el botón de *Go back home*. Ahora si volvemos a la lista de los tests de escucha, no podemos volver a entrar en él ya que nos aparece como hecho.



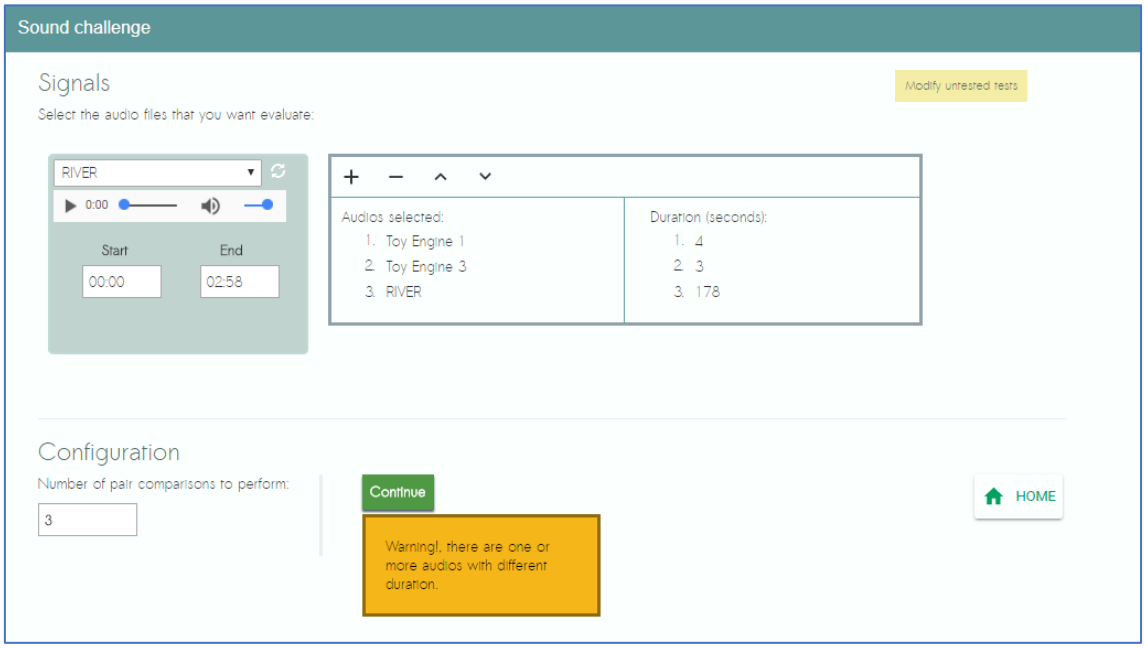
Creación, modificación y análisis de los tests

Este apartado está orientado a los diseñadores de las pruebas de escucha, para cederles una guía rápida de cómo gestionar los test siendo administrador.

1. Crear un test desde cero: Entrar a la aplicación como antes se ha explicado y clicar en la pestaña de arriba de *Design listening tests* y dentro de ella clicar en *create and modify*:



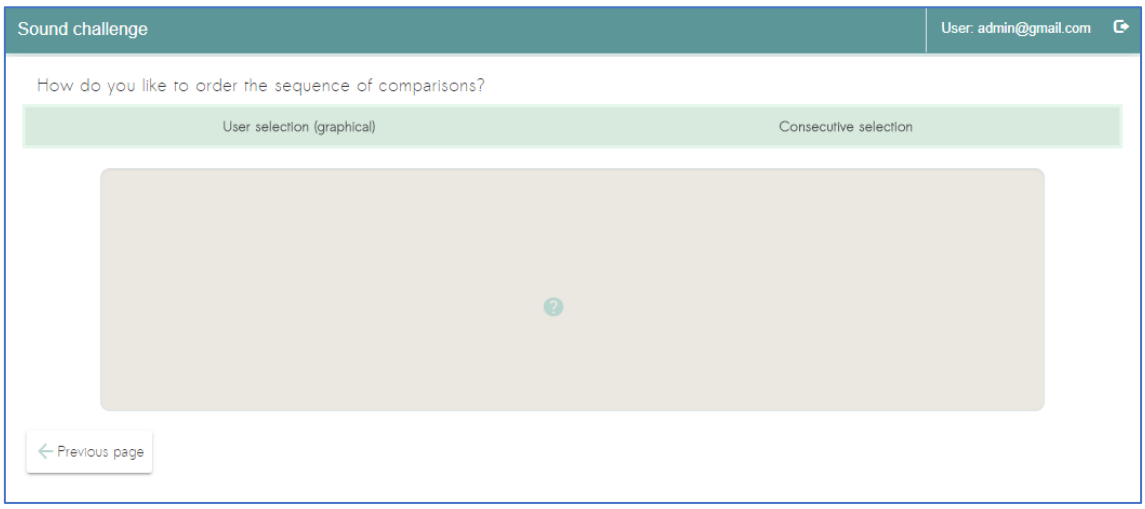
2. La siguiente pantalla permite seleccionar los audios de la base de datos, añadirlos y decidir que parte del archivo de audio se desea añadir en el test (*starr* y *end*). Además de seleccionar el número de pares a diseñar.

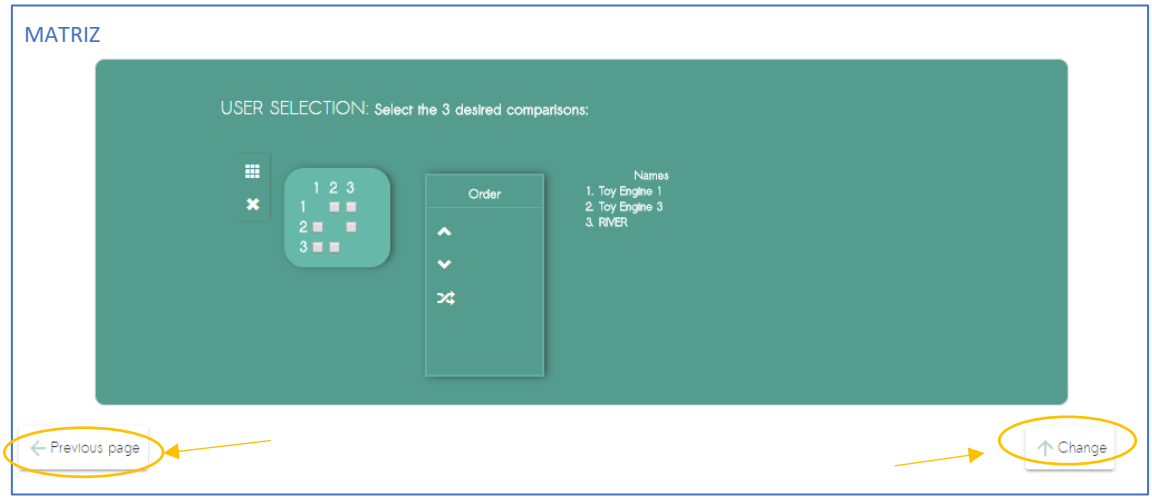


Si introduces audios con diferente duración te aparecerá el *warning* anterior pero te dejará continuar bajo tu responsabilidad, pero si directamente un audio tiene duración 0, no dejará avanzar.

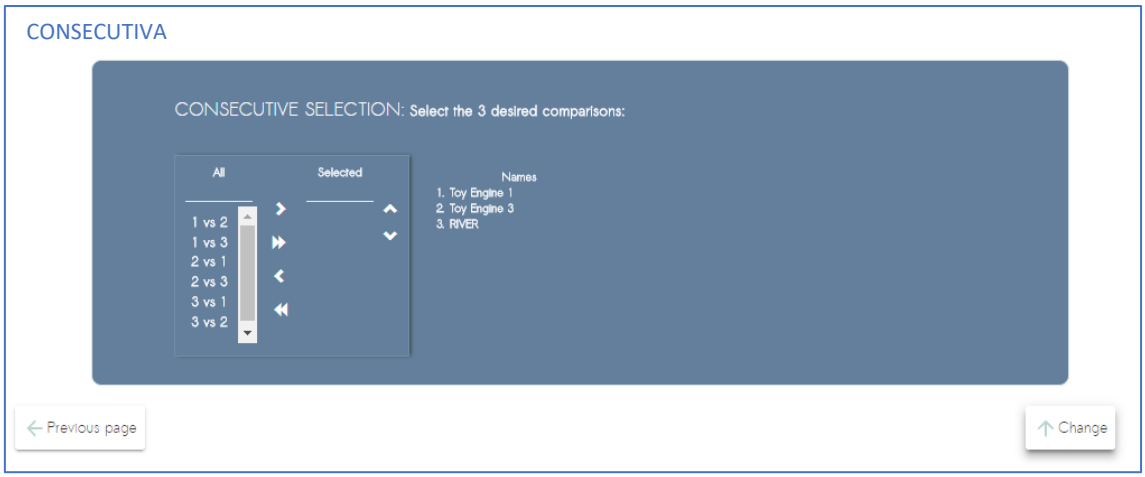
3. Si todo está correcto y clicamos a *Continue*, nos llevará a la pantalla de selección de pares, de umbral y de características a evaluar en la prueba.

Primero hay que seleccionar cómo se van a elegir los pares de comparaciones, hay dos formas: en matriz o de forma consecutiva.

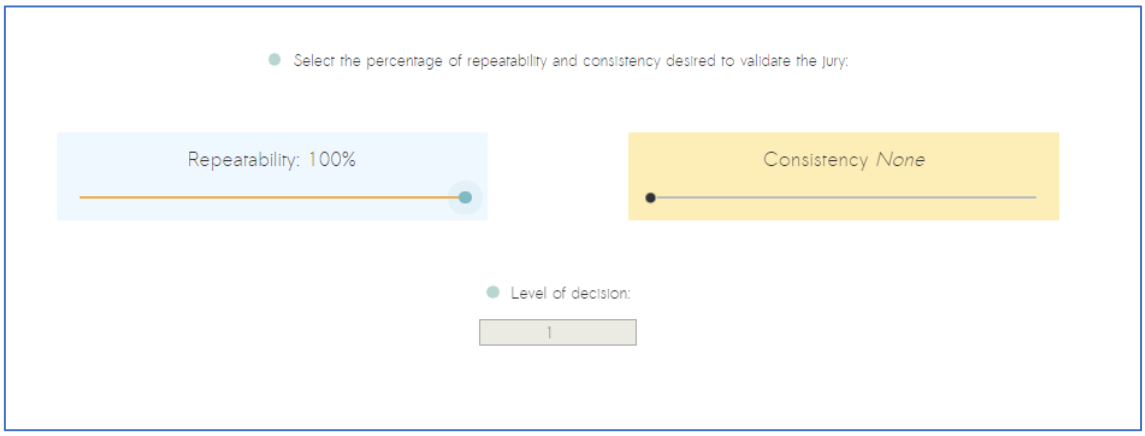




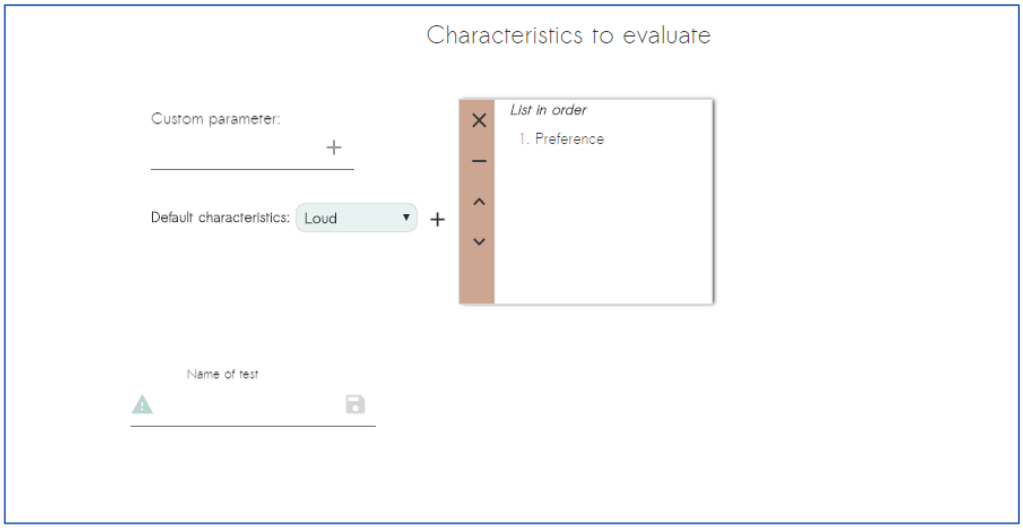
Se puede cambiar el método con el botón *change* y si se desea volver a la selección de audios se debe clicar en *previous page*.



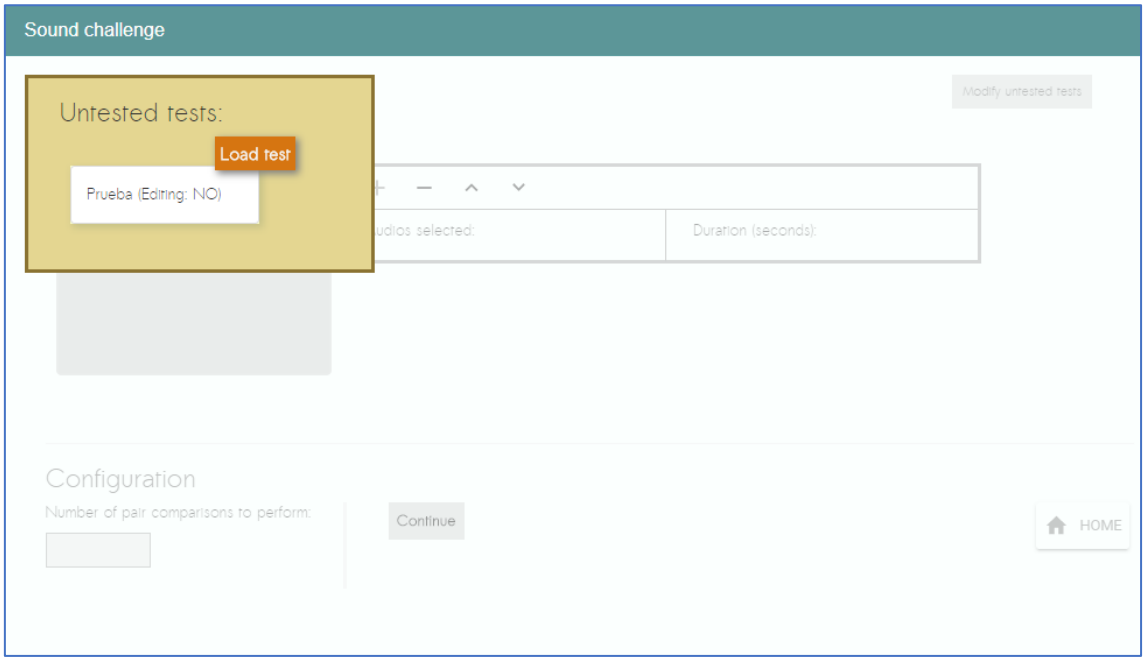
Seleccionamos por tanto el umbral de repetitividad y el valor de consistencia ideal una vez TODOS los pares han sido seleccionados. A veces ocurre que no hay suficientes pares para que exista consistencia, a priori no es un gran problema.



Por último se añaden las características a evaluar y el nombre del test. Si todo está correcto te permite guardar el test.

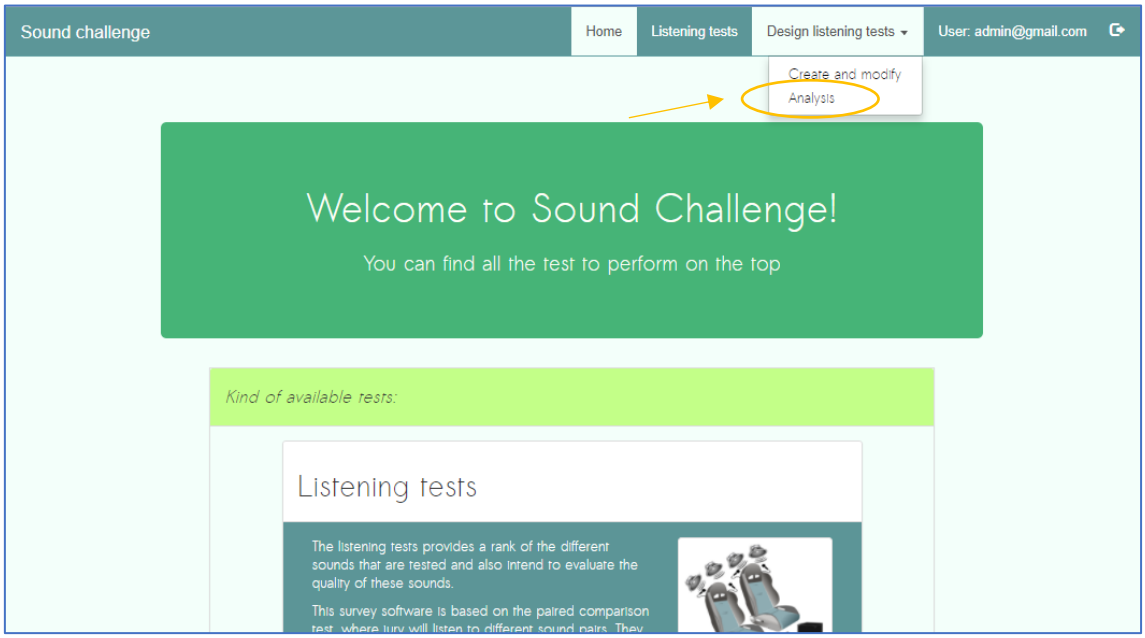


4. Si se desea modificar algún test no realizado por ningún usuario se va al mismo apartado de crear el test desde cero y se clica en el botón de *Modify untested tests* apareciendo si los hubiera algún test en esas condiciones.

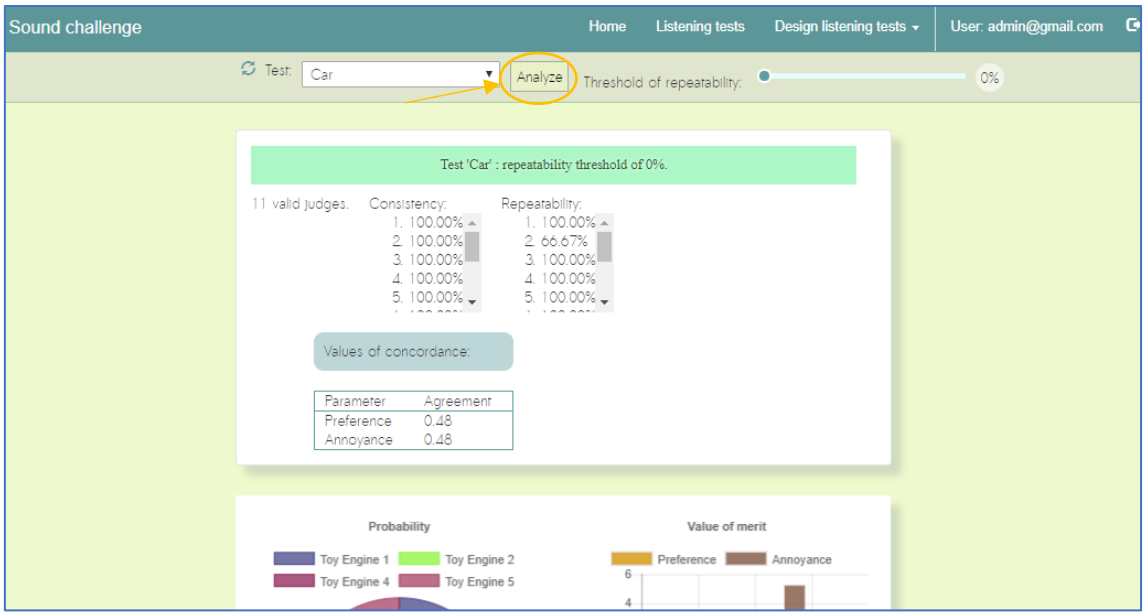


El estado de *EDITING* indica si se está editando. Cuando esté en 'YES' no se podrá ejecutar el test. Este estado vuelve a 'NO' cuando se vuelve a guardar con el mismo nombre en la ventana de configuración de pares. Para cargar el test simplemente se selecciona y se clica en *load test*.

5. Analizar tests: Seleccionamos esta vez la opción de *Analysis.*



Aparecerá la siguiente pantalla donde se debe seleccionar un test de la lista y si tiene usuarios válidos con respecto al umbral de repetitividad (*threshold of repetability*) sacará los datos.


Sound challenge
Home
Listening tests
Design listening tests
User: admin@gmail.com
Test:
Car
Analyze
Threshold of repeatability:
0%
Test 'Car' : repeatability threshold of 0%.
11 valid judges.
Consistency:
1. 100.00%
2. 100.00%
3. 100.00%
4. 100.00%
5. 100.00%
Repeatability:
1. 100.00%
2. 66.67%
3. 100.00%
4. 100.00%
5. 100.00%
Values of concordance:
Parameter
Agreement
Preference
0.48
Annoyance
0.48
Probability
Toy Engine 1
Toy Engine 2
Toy Engine 4
Toy Engine 5
Value of merit
Preference
Annoyance
6
4

ANEXO 2: Código JavaScript del análisis

```
// FUNCTION to analyze the test results

var btn_ana = document.getElementById('btn_ana');
sel_ana.onchange = function(){


    get_login_tests(function(tests,users_in_test,logins){
        var cnt = 0;
        for(var i=0;i< users_in_test.length; i++){
            if(parseInt(sel_ana.options[sel_ana.selectedIndex].id) ===
                                    users_in_test[i]['fields']['test']){
                cnt+=1;
            }
        }
        if(cnt>0){

            // FIRST: Exclude the jury that don't have enough repeatability
            for (i = 0; i < tests.length; i++) {
                if (parseInt(sel_ana.options[sel_ana.selectedIndex].id) === tests[i]['pk']) {
                    var test_selected = tests[i];
                    break;
                }
            }

            // Get the data of design
            var data_test_aux = test_selected['fields']['sound_data'];
            var data_test = JSON.parse(data_test_aux.replace(/&quot;/g, '"'));
            // ------- Analize the data from the test--------//

            // Thresholds of consistency and repetitiveness of design
            var th_cons = data_test[0]['Consistency']; //Integer, i.e.: 0, 50, 100
            var th_rep = data_test[0]['Repetitiveness']['Value']; //Integer, i.e.: 0, 50, 100
            var th_rep_step = data_test[0]['Repetitiveness']['Step']; //Integer, i.e.: 0, 50, 100

            document.getElementById('th_rep').disabled = false;
            document.getElementById('th_rep').value=th_rep;
            document.getElementById('th_rep').setAttribute('step',th_rep_step);
            document.getElementById('text_rep').style.display = 'inline-block';
            document.getElementById('text_rep').innerText = th_rep + '%';

            btn_ana.disabled = false;
            btn_ana.onclick= function(){



                var content = document.getElementById('ana_content');
                var child_content = content.children;
                if(child_content.length >0){
                    for(i=child_content.length-1; i>=0; i--){
                        content.removeChild(child_content[i]);
                    }
                }
                var th_rep_sel =  document.getElementById('th_rep').value;
                var valid_users = [];
                var cnt_val = 0;
                for (i = 0; i < users_in_test.length; i++) {
                    if (parseInt(sel_ana.options[sel_ana.selectedIndex].id) ===
                                        users_in_test[i]['fields']['test']){
                        var data_aux = users_in_test[i]['fields']['data'];
                        var data = JSON.parse(data_aux.replace(/&quot;/g, '"'));
                        var cons = data['Analysis data']['Consistency'];
                        var rep = data['Analysis data']['Repetitiveness'];
                        if (rep >= th_rep_sel) {
                            valid users[cnt val] = users in test[i]; // Valid jury to make analysis
                            cnt_val += 1;
                        }
                    }

                }
```

```
                if(valid_users.length <1){
                    alert("There aren't valid users");
                }else{

                    if(child_content.length >0){
                        for(i=child_content.length-1; i>=0; i--){
                            content.removeChild(child content[i]);
                        }
                    }
                    var results_param = [];
                    results_param[0] = []; // Concord
                    results_param[1] = []; // Value of merit
                    results_param[2] = []; // Probability
                    results param= analyze param(test selected,valid users);


                    // Show the graphs
                    make_graph(th_rep_sel,results_param,test_selected,valid_users,content,logins)

                }

            };
        }else{
            // Don't have any user in the test
            btn_ana.disabled = true;
            document.getElementById('th_rep').disabled = true;
            document.getElementById('th_rep').value = 0;
            document.getElementById('text_rep').style.display = 'none';
        }
    });

};

// Function to analyze the parameters

function analyze_param(test,valid_users){


    var data_test_aux = test['fields']['sound_data'];
    var data_test = JSON.parse(data_test_aux.replace(/&quot;/g, '"'));
    // Number of signals:
    var nsignals = data_test[0]['Number_sounds'];
    // Matrix of comparisons pairs
    var MatrixComp = new Array(data_test.length);
    for(var i=0; i< data_test.length; i++){
        MatrixComp[i] = new Array(2);
        MatrixComp[i][0] = parseInt(data_test[i]['First_sound']['sound_index']);
        MatrixComp[i][1] = parseInt(data_test[i]['Second_sound']['sound_index']);
    }
    var ncomp = data_test.length;

    // Initialization

    // Valid number of jurys on test:
    var nJury = valid_users.length;

    // Extract whole results for that parameter
    var data_par_aux = test['fields']['par_data']; // Data of parameters to evaluate
    var data_par = JSON.parse(data_par_aux.replace(/&quot;/g, '"'));


    var results = new Array(data_par.length);
    var pref = new Array(data_par.length); // Variable of preference
    var out = new Array(data_par.length);
    var ponder  = new Array(data_par.length);
    var alfa  = new Array(data_par.length);
    var sum_1 = new Array(data_par.length);
    var sum_2 = new Array(data_par.length);

    // Initialization of sumarize
    for(l=0; l<data_par.length; l++) {
        sum_1[l] =new Array(ncomp);
        sum_2[l] =new Array(ncomp);
        for (j = 0; j < ncomp; j++) {
            sum_1[l][j] = 0;
```

```
            sum_2[l][j] = 0;

        }
    }
    // Matrix of results
    for(var l=0; l<data_par.length; l++){ // Each array is a parameter, the first one is the
preference
        results[l] =new Array(nJury);

        for(i=0; i<nJury;i++){
            results[l][i] = new Array(ncomp);
            var data_user_aux = valid_users[i]['fields']['data'];
            var data_user = JSON.parse(data_user_aux.replace(/&quot;/g, '"'));
            for(var j=0; j<ncomp;j++){
                results[l][i][j]= data_user['Choice data'][j][l];
                // Sumarize the times that the users prefer one sound or another for each parameter
                if (results[l][i][j] === 1) {
                    sum_1[l][j] = sum_1[l][j] + 1;

                }
                if (results[l][i][j] === 2) {
                    sum_2[l][j] = sum_2[l][j] + 1;
                }
            }
        }
    }

    for(l=0; l<data_par.length; l++) {
        pref[l] = []; // Variable of preference
        out[l] = [];
        ponder[l] = [];
        alfa[l] = [];
        for (i = 0; i <nsignals; i++) {
            pref[l][i] = [];
            out[l][i] = [];
            ponder[l][i] = [];
            alfa[l][i] = [];
            for(j=0; j<nsignals; j++){
                pref[l][i][j] = 0;
                out[l][i][j] = 0;
                ponder[l][i][j] = 0;
                alfa[l][i][j] = 0;
            }
        }
    }


    // Matrix of preference for each parameter with the times the user prefers the signal for each
parameter
    for(l=0; l<data_par.length; l++){
        for(var k=0; k<ncomp; k++) {
            var first = MatrixComp[k][0]-1;
            var second = MatrixComp[k][1]-1;
            pref[l][first][second] = pref[l][first][second] + sum_1[l][k];
            pref[l][second][first] = pref[l][second][first] + sum_2[l][k];
        }
    }



    alfa = pref.slice();

    var  tras_alfa = [];
    for(l=0; l<data_par.length; l++) {
        tras_alfa[l] = [];
        for (i = 0; i < nsignals; i++) {
            tras_alfa[l][i] = [];
            for (j = 0; j < nsignals; j++) {
                tras_alfa[l][i][j] = alfa[l][j][i];
            }
        }
    }

    for(l=0; l<data_par.length; l++) {
        for (i = 0; i < nsignals; i++){
            for(j=0; j< nsignals; j++) {
```

```
                ponder[l][i][j]= alfa[l][i][j]+tras alfa[l][i][j];

                if(ponder[l][i][j] === 0){
                    ponder[l][i][j] = 1;
                }

                out[l][i][j] = pref[l][i][j]/ponder[l][i][j];

            }

        }
    }

    // Concordance coefficient for each parameter
    var concord =new Array(data_par.length);
    concord  = coef_concor(results,nJury,ncomp);

    var vmerit = new Array(data_par.length);
    vmerit = valoration(out,nsignals);

    for(i=0; i<data_par.length; i++){
        if(data_par[i]=== 'Preference'){
            var indexPref = i;
            break;
        }
    }

    var prob= new Array(nsignals);
    prob = calc_prob(alfa[indexPref], MatrixComp,nsignals,nJury);


    var results_param = [];
    results_param[0] = concord;
    results_param[1] = vmerit;
    results_param[2] = prob;
    results_param[3] = indexPref;

    return results_param;


}

// Function to calculate the concordance
function coef_concor(results, nJury,ncomp){
    var concord = new Array(results.length);

    //Number of times juries agree
    var Z = new Array(results.length);
    {#                var cnt = 0;#}
    for( var l=0; l< results.length; l++){
        Z[l] = 0;
        for(var i=0; i<nJury; i++) {
            for (var j = i + 1; j < nJury; j++) {
                for(var k=0; k<ncomp; k++){
                    if(results[l][i][k] === results[l][j][k]){
                        Z[l]+=1;
                    }
                }
            }
        }
        concord[l] = (8*Z[l]/(nJury*(nJury-1)*2*ncomp))-1;
    }

    return concord;
}




// Function that returns the subjective values of the parameter whose probability matrix is p
function valoration(p,nsignals){
```

```
    var d = new Array(p.length);
    var pro = new Array(p.length);
    var V = new Array(p.length);
    var sum = new Array(p.length);

    for (var l=0; l<p.length; l++){
        d[l] = [];
        for(var i=0; i<nsignals; i++){
            d[l][i] = [];
            for(var j=0; j<nsignals; j++){
                d[l][i][j] = 0;
            }
        }
    }

    for (l=0; l<p.length; l++){
        sum[l] = [];
        pro[l]=[];
        V[l] = [];
        for(i=0; i< nsignals;i++){
            pro[l][i]=[];
            sum[l][i]=0;
            for(j=i+1;j<nsignals;j++){
                pro[l][i][j]=p[l][i][j];

                if(pro[l][i][j] === 1){
                    pro[l][i][j] = 0.999;
                }
                if(pro[l][i][j] === 0){
                    pro[l][i][j] = 0.001;
                }

                d[l][i][j] = 2*Math.atanh(2*pro[l][i][j]-1);
                d[l][j][i] =d[l][i][j] * (-1);
                // d is OK

            }
            for(var k=0; k<nsignals;k++){ // Sum the rows
                sum[l][i] = sum[l][i]+ d[l][i][k];
            }
            V[l][i]=sum[l][i]/(nsignals-1);
        }

    }

    return V;
}

// Function to calculate the probability of choosing each signal
function calc_prob(alfa, compa, nsignals, nJury){

    var Nij = new Array(nsignals);
    var Nij_tras = new Array(nsignals);
    var vectorp_o = new Array(nsignals);
    var vectorp_i = new Array(nsignals);
    var Wi = new Array(nsignals);
    var sum_total =1;
    var sum_vectorp_o = 0;
    for(var i=0; i<nsignals; i++){
        Nij[i] = [];
        Nij_tras[i]=[];
        // Initialize final probability vector and initial
        vectorp_o[i]= Math.random();
        vectorp i[i]= -1;
        Wi[i]= 0;
        sum vectorp o+= vectorp o[i];
        for(var j=0; j<nsignals; j++){
            Nij[i][j] = 0;
            Nij tras[i][j] = 0;
        }
    }

    for(i=0; i<compa.length; i++){
        Nij[compa[i][0]-1][compa[i][1]-1] = nJury;
    }
```

```
    for(i=0; i<nsignals; i++){
        vectorp o[i] = vectorp o[i]/sum vectorp o;
        for(j=0; j<nsignals; j++){
            // Calculate Wi number of times i is preferred to j
            Wi[i] += alfa[i][j];

            Nij tras[j][i]=Nij[i][j];
        }
    }


    for(i=0; i<nsignals; i++){
        for(j=0; j<nsignals; j++){
            // Number of comparisons  between i and j is the same as for j and i
            Nij[i][j]=Nij[i][j] + Nij_tras[i][j];
        }
    }


    var sum_2 = 0;
    while(sum_total > 0.001){
        sum_total = 0;
        for (i = 0; i < nsignals; i++) {
            vectorp i[i] = vectorp o[i];
        }
        for (var l = 0; l < nsignals; l++) {
            // Denominator
            var Den = 0;
            for (j = 0; j < nsignals; j++) {
                if (j !== l) {
                    Den = Den + Nij[l][j]/(vectorp_i[l] + vectorp_o[j]);
                }
            }

            vectorp o[l] = Wi[l] / Den;
        }

        // Normalize vectorp_o to be probabilities
        for (var k = 0; k < nsignals; k++) {
            sum 2 += vectorp o[k];
        }

        for (var h = 0; h < nsignals; h++) {
            vectorp_o[h] = vectorp_o[h] / sum_2;
        }
        sum 2 = 0;
        for (var p = 0; p < nsignals; p++) {
            var aux = vectorp_o[p] - vectorp_i[p];
            sum_total += Math.pow(aux,2);
        }

    }


    return vectorp_o;


}

// Function to graph the data of analysis
function make_graph(th,results_param,test_selected,valid_users,DIVcontent, logins){
    //results_param[0] = concordance (dimension = number of parameters)
    //results param[1] = values of merit (dimension = number of parameters)
    //results param[2] = probability (dimension = number of signals)
    //results_param[3] = index of preference (dimension = 1)




    var DIV_data = document.createElement('div');
    DIV_data.className = 'DIV_ana well';
    DIV_data.style.minWidth= '380x';
    DIV_data.style.marginBottom= '30px';
```

```
    DIV_data.style.marginTop= '40px';
    DIV_data.style.maxWidth= '661px';



    var DIV_prob = document.createElement('div');
    DIV_prob.style.float='left';
    DIV_prob.className='DIV_ana';

    var DIV_vmerit = document.createElement('div');
    DIV_vmerit.style.float='left';
    DIV_vmerit.className='DIV_ana';

    var DIV_conc = document.createElement('div');
    DIV_conc.style.display = 'inline-block';
    DIV_conc.style.marginLeft = '10%';


    var sounds_aux = test_selected['fields']['list_of_sounds'];
    var sounds = JSON.parse(sounds_aux.replace(/&quot;/g, '"'));
    var par_aux = test_selected['fields']['par_data'];
    var par = JSON.parse(par_aux.replace(/&quot;/g, '"'));
    var dataTest_aux = test_selected['fields']['sound_data'];
    var dataTest = JSON.parse(par_aux.replace(/&quot;/g, '"'));

    var threshold = document.createElement('div');
    threshold.style.width='97%';
    threshold.style.background = '#adf7c6';
    threshold.style.padding = '10px';
    threshold.style.marginBottom = '10px';
    threshold.style.textAlign = 'center';
    threshold.style.color = '#265f2d';
    threshold.style.fontFamily = 'sans serif';
    threshold.style.textShadow = '#ffffff 0px 0px 5px';
    threshold.innerText = 'Test' + " '"+ test_selected['fields']['name'] + "' "+ ': repeatability
threshold of '+ th + '%.';

    var concordance_table = document.createElement('table');
    {#                concordance_table.style.margin = '0 auto';#}
    concordance_table.style.padding = '10px';
    concordance_table.style.border = 'solid thin cadetblue';
    var tr = document.createElement('tr');
    var th1 = document.createElement('th');
    var th2 = document.createElement('th');

    th1.innerText = 'Parameter';
    th1.style.borderBottom = 'solid thin cadetblue';
    th2.style.borderBottom = 'solid thin cadetblue';
    th1.className = 'th_conc';
    th2.className = 'th_conc';
    th2.innerText = 'Agreement';

    tr.appendChild(th1);
    tr.appendChild(th2);
    concordance_table.appendChild(tr);

    for(i=0; i< par.length; i++){
        tr = document.createElement('tr');
        th1 = document.createElement('th');
        th2 = document.createElement('th');

        th1.className = 'th_conc';
        th2.className = 'th_conc';
        th1.innerText = par[i];
        th2.innerText = (results_param[0][i]).toFixed(2);

        tr.appendChild(th1);
        tr.appendChild(th2);
        concordance table.appendChild(tr);
    }



    var labels = [];
```

```
    for(var i=0; i<sounds.length; i++){
        labels[i] = sounds[i]['Soundname'];
    }


    var data_prob = [];
    var color_prob = [];
    var data_vmerit= [];

    var border_color_prob = [];
    for(i=0; i<sounds.length; i++){
        data_prob[i] =(results_param[2][i] * 100).toFixed(2) ;
        var r = Math.trunc(Math.random()*255);
        var g = Math.trunc(Math.random()*255);
        var b = Math.trunc(Math.random()*255);

        color_prob[i] = 'rgba('+r+', '+g+', '+b+', 0.7)';
        border_color_prob[i] = 'rgba('+r+', '+g+', '+b+', 1)';

    }


    for(var j=0;j<par.length; j++){
        var r1 = Math.trunc(Math.random()*255);
        var g1 = Math.trunc(Math.random()*255);
        var b1 = Math.trunc(Math.random()*255);
        data_vmerit[j]= new Data_const(par[j],'rgba('+r1+', '+g1+', '+b1+',
0.8)',results_param[1][j]);
    }


    var cons_jury = document.createElement('div');
    cons_jury.innerText = 'Consistency:';
    cons_jury.style.display = 'inline-block';
    cons_jury.style.float = 'left';
    cons_jury.style.marginLeft = '20px';
    cons_jury.style.overflow = 'auto';
    var rep_jury = document.createElement('div');
    rep_jury.innerText = 'Repeatability:';
    rep_jury.style.display = 'inline-block';
    rep_jury.style.float = 'left';
    rep_jury.style.marginLeft = '20px';
    rep_jury.style.overflow = 'auto';


    var num_jury = document.createElement('div');
    num_jury.innerText = valid_users.length + ' valid judges.';
    num_jury.style.minWidth='50px';
    num_jury.className='num_Jury';
    num_jury.style.wordWrap='break-word';
    num_jury.style.display = 'inline-block';
    num_jury.style.float = 'left';
    num_jury.style.overflow = 'auto';

    {#                var list_juries = document.createElement('ol');#}
    var list_juries_cons = document.createElement('ol');
    var list_juries_rep = document.createElement('ol');
    var valid_data =[];
    var flag_valid =0;

    for(i=0; i<valid_users.length; i++){
        var aux_valid_data = valid_users[i]['fields']['data'];
        valid_data[i] = JSON.parse(aux_valid_data.replace(/&quot;/g, '"'));
        var user = valid_users[i]['fields']['email'];
        {#                    var li = document.createElement('li');#}
        var li2 = document.createElement('li');
        var li3 = document.createElement('li');
        var aux_co = valid_data[i]['Analysis data']['Consistency'];
        if(aux_co!== 'None'){
            li2.innerText = (aux_co).toFixed(2) + '%';
        }else{
            li2.innerText = aux_co;
        }

        var aux_re = valid_data[i]['Analysis data']['Repetitiveness'];
        li3.innerText = (aux_re).toFixed(2) + '%';
```

```
        for(j=0; j< logins.length; j++){
            if(logins[j]['fields']['username_email'] === user){
                var name = logins[j]['fields']['username_email'];
                flag valid = 0;
            }else{
                flag_valid +=1;
            }
        }
        if(flag_valid === logins.length){
            name = user + ": Doesn't exist!";
            li2.title = "Warning: This user doesn't exist on the Registered User Database.";
            li3.title = "Warning: This user doesn't exist on the Registered User Database.";
            li2.style.color = 'red';
            li3.style.color = 'red';
            li2.style.fontStyle = 'italic';
            li3.style.fontStyle = 'italic';


        list_juries_cons.className = 'list_users';
        list_juries_rep.className = 'list_users';
        list juries cons.appendChild(li2);
        list_juries_rep.appendChild(li3);
        flag_valid=0;
    }


    var div_text = document.createElement('div');

    div_text.innerText = 'Values of concordance: ';
    div_text.style.background = 'rgba(95, 158, 160, 0.41)';
    div_text.style.padding = '10px';
    div_text.style.borderRadius = '10px';
    div_text.style.marginTop = '10px';
    div_text.style.marginBottom = '20px';

    DIV conc.appendChild(div text);
    DIV conc.appendChild(concordance table);
    cons jury.appendChild(list juries cons);
    rep_jury.appendChild(list_juries_rep);
    DIV_data.appendChild(threshold);
    DIV_data.appendChild(num_jury);
    DIV data.appendChild(cons jury);
    DIV data.appendChild(rep jury);
    DIV_data.appendChild(DIV_conc);
    DIVcontent.appendChild(DIV_data);

    var graph_prob = document.createElement('canvas');
    graph_prob.id='chart_prob';
    graph_prob.setAttribute('width','300');
    graph_prob.setAttribute('height','300');
    DIV_prob.style.margin='0 auto';
    DIV prob.appendChild(graph prob);
    DIVcontent.appendChild(DIV prob);


    var graph_vmerit = document.createElement('canvas');
    graph_vmerit.id='chart_vmerit';
    graph_vmerit.setAttribute('width','300');
    graph_vmerit.setAttribute('height','300');
    DIV_vmerit.style.margin='0 auto';
    DIV_vmerit.appendChild(graph_vmerit);
    DIVcontent.appendChild(DIV_vmerit);

    var probability = new Chart(graph_prob, {
        type: 'pie',
        data: {
            labels: labels,
            datasets: [{
                label: '% Probability',
                data: data_prob,
                backgroundColor: color_prob,
                borderColor: border_color_prob ,
                borderWidth: 1
            }]
        },
        options: {
```

```
            title:{
                display: true,
                text: 'Probability'
            },
            animation:{
                animateScale:true
            }
        }
    });
    var vmerit = new Chart(graph_vmerit, {
        type: 'bar',
        data: {
            labels: labels,
            datasets: data_vmerit
        },
        options:{
            title:{
                display: true,
                text: 'Value of merit'
            }
        }
    });
}

function Data_const(par,color,res){
    this.label= par;
    this.backgroundColor= color;
    this.data = res;
}
```